

O'REILLY®

3-е издание



Data Mining

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ
ИЗ FACEBOOK, TWITTER, LINKEDIN, INSTAGRAM, GITHUB

 ПИТЕР®

Мэтью Рассел
Михаил Классен

THIRD EDITION

Mining the Social Web

Matthew A. Russell and Mikhail Klassen

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY[®]

Мэтью Рассел
Михаил Классен

Data Mining

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ
ИЗ FACEBOOK, TWITTER,
LINKEDIN, INSTAGRAM, GITHUB



Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону
Самара • Минск

2020

ББК 32.973.233-018+32.988.02

УДК 004.62+004.738.5

P24

Рассел Мэтью, Классен Михаил

P24 Data Mining. Извлечение информации из Facebook, Twitter, LinkedIn, Instagram, GitHub. — СПб.: Питер, 2020. — 464 с.: ил. — (Серия «IT для бизнеса»).

ISBN 978-5-4461-1246-3

В недрах популярных социальных сетей — Twitter, Facebook, LinkedIn и Instagram — скрыты богатейшие залежи информации. Из этой книги исследователи, аналитики и разработчики узнают, как извлекать эти уникальные данные, используя код на Python, Jupyter Notebook или контейнеры Docker.

Сначала вы познакомитесь с функционалом самых популярных социальных сетей (Twitter, Facebook, LinkedIn, Instagram), веб-страниц, блогов и лент, электронной почты и GitHub. Затем приступите к анализу данных на примере Twitter.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.233-018+32.988.02

УДК 004.62+004.738.5

Права на издание получены по соглашению с O'Reilly. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-1491985045 англ.

Authorized Russian translation of the English edition of Mining the Social Web, 3rd Edition. ISBN 9781491985045 © 2019 Matthew A. Russell, Mikhail Klassen. This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

ISBN 978-5-4461-1246-3

© Перевод на русский язык ООО Издательство «Питер», 2020

© Издание на русском языке, оформление ООО Издательство «Питер», 2020
© Серия «IT для бизнеса», 2020

Краткое содержание

Предисловие 16

ЧАСТЬ I. ЭКСКУРСИЯ ПО СОЦИАЛЬНЫМ СЕТЯМ

Вступление 32

Глава 1. Twitter: исследование актуальных тем, о чем говорят люди
и многое другое..... 34

Глава 2. Facebook: анализ фан-страниц, исследование дружественных связей
и многое другое..... 79

Глава 3. Instagram: компьютерное зрение, нейронные сети, распознавание
объектов и лиц 126

Глава 4. LinkedIn: классификация по профессиям, группировка коллег и многое
другое 161

Глава 5. Анализ текстовых файлов: определение сходства документов,
извлечение словосочетаний и многое другое..... 208

Глава 6. Анализ веб-страниц: использование методов обработки
естественного языка, обобщение статей из блогов и многое другое 251

Глава 7. Анализ электронной почты: кто кому пишет, о чем, как часто и многое
другое 305

Глава 8. Анализ GitHub: особенности сотрудничества при разработке ПО,
графы интересов и многое другое 344

ЧАСТЬ II. СБОРНИК РЕЦЕПТОВ ДЛЯ TWITTER

Глава 9. Сборник рецептов для Twitter	394
--	-----

ЧАСТЬ III. ПРИЛОЖЕНИЯ

Приложение А. Информация о виртуальной машине с примерами для этой книги.....	452
Приложение Б. Основы OAuth.....	454
Приложение В. Советы и рекомендации для Python и Jupyter Notebook	460
Об авторах	461
Об обложке	462

Оглавление

Предисловие	16
Примечание Мэтью Рассела	16
README.1st	17
Предвосхищая ожидания	17
Технологии на основе Python	20
Новое в третьем издании.....	22
Этические аспекты добычи данных.....	24
Типографские соглашения.....	26
Использование примеров программного кода	27
Благодарности к третьему изданию	28
Благодарности ко второму изданию.....	28
Благодарности к первому изданию	29
От издательства	30
 ЧАСТЬ I. ЭКСКУРСИЯ ПО СОЦИАЛЬНЫМ СЕТЯМ.....	31
Вступление	32
 Глава 1. Twitter: исследование актуальных тем, о чем говорят люди и многое другое.....	34
1.1. Обзор	34
1.2. Причины популярности Twitter	35
1.3. Twitter API.....	38
1.3.1. Базовая терминология Twitter	38
1.3.2. Подключение к Twitter API	41
1.3.3. Исследование актуальных тем	46
1.3.4. Поиск твитов	51

1.4. Анализ 140 (или более) символов	58
1.4.1. Извлечение сущностей из твита.....	60
1.4.2. Исследование твитов и сущностей в них с применением частотного анализа	62
1.4.3. Определение лексического разнообразия твитов.....	65
1.4.4. Исследование шаблонов в ретвитах.....	68
1.4.5. Визуализация частот с помощью гистограмм	71
1.5. Заключительные замечания.....	75
1.6. Упражнения	76
1.7. Онлайн-ресурсы	78
Глава 2. Facebook: анализ фан-страниц, исследование дружественных связей и многое другое.....	79
2.1. Обзор	80
2.2. Facebook Graph API	81
2.2.1. Знакомство с Graph API.....	83
2.2.2. Знакомство с Open Graph Protocol	88
2.3. Анализ связей в социальном графе.....	96
2.3.1. Анализ страниц в Facebook	100
2.3.2. Манипулирование данными с помощью pandas.....	113
2.4. Заключительные замечания.....	122
2.5. Упражнения.....	123
2.6. Онлайн-ресурсы.....	124
Глава 3. Instagram: компьютерное зрение, нейронные сети, распознавание объектов и лиц	126
3.1. Обзор	127
3.2. Instagram API.....	128
3.2.1. Выполнение запросов к Instagram API	129
3.2.2. Извлечение своей ленты постов из Instagram	132
3.2.3. Извлечение медиафайлов по хештегу.....	134
3.3. Анатомия поста в Instagram	135
3.4. Краткое введение в искусственные нейронные сети	138
3.4.1. Обучение нейросети «рассматриванию» изображений.....	140
3.4.2. Распознавание рукописных цифр.....	142
3.4.3. Распознавание объектов на фотографиях с помощью предварительно обученных нейросетей	148

3.5. Применение нейронных сетей для анализа постов в Instagram	152
3.5.1. Классификация содержимого изображения.....	154
3.5.2. Определение лиц на изображениях	154
3.6. Заключительные замечания	156
3.7. Упражнения	157
3.8. Онлайн-ресурсы.....	158
Глава 4. LinkedIn: классификация по профессиям, группировка коллег и многое другое.....	161
4.1. Обзор	162
4.2. LinkedIn API	163
4.2.1. Выполнение запросов к LinkedIn API.....	163
4.2.2. Загрузка файла с информацией о контактах в LinkedIn	168
4.3. Краткое введение в приемы кластеризации данных.....	169
4.3.1. Нормализация данных для анализа.....	171
4.3.2. Измерение степени сходства.....	185
4.3.3. Алгоритмы кластеризации	188
4.4. Заключительные замечания	204
4.5. Упражнения	205
4.6. Онлайн-ресурсы.....	206
Глава 5. Анализ текстовых файлов: определение сходства документов, извлечение словосочетаний и многое другое.....	208
5.1. Обзор	209
5.2. Текстовые файлы.....	209
5.3. Краткое введение в TF-IDF	211
5.3.1. Частота слова	212
5.3.2. Обратная частота документа	214
5.3.3. TF-IDF.....	216
5.4. Оценка запросов данных на естественном языке с использованием TF-IDF... ..	220
5.4.1. Введение в Natural Language Toolkit	221
5.4.2. Вычисление оценки TF-IDF для текста на естественном языке.....	225
5.4.3. Поиск похожих документов.....	227
5.4.4. Анализ биграмм на естественном языке.....	234
5.4.5. Размышления об анализе данных на естественном языке	246
5.5. Заключительные замечания.....	248

5.6. Упражнения	249
5.7. Онлайн-ресурсы	250
Глава 6. Анализ веб-страниц: использование методов обработки естественного языка, обобщение статей из блогов и многое другое	251
6.1. Обзор	252
6.2. Скрапинг, парсинг и обход сайтов в интернете	253
6.2.1. Обход страниц методом поиска в ширину	257
6.3. Определение семантики декодированием синтаксиса	261
6.3.1. Пошаговая иллюстрация обработки естественного языка	264
6.3.2. Выделение предложений из данных на человеческом языке	268
6.3.3. Обобщение документов	273
6.4. Анализ сущностей: смена парадигмы	286
6.4.1. Определение общего смысла данных на человеческом языке	291
6.5. Оценка качества при анализе данных на человеческом языке	297
6.6. Заключительные замечания	300
6.7. Упражнения	301
6.8. Онлайн-ресурсы	303
Глава 7. Анализ электронной почты: кто кому пишет, о чем, как часто и многое другое	305
7.1. Обзор	307
7.2. Получение и обработка корпуса с почтовыми сообщениями	307
7.2.1. Пример почтового ящика UNIX	307
7.2.2. Получение корпуса Enron	313
7.2.3. Преобразование почтового корпуса в формат mbox	316
7.2.4. Преобразование почтовых ящиков UNIX в объекты DataFrames	317
7.3. Анализ корпуса Enron	320
7.3.1. Запрос по диапазону времени	322
7.3.2. Анализ закономерностей во взаимодействиях отправителей и получателей	325
7.3.3. Поиск писем по ключевым словам	330
7.4. Анализ собственных почтовых данных	332
7.4.1. Доступ к почтовому ящику Gmail с использованием OAuth	334
7.4.2. Извлечение и парсинг электронных писем	337
7.4.3. Визуализация закономерностей в электронных письмах с помощью Immersion	339

7.5. Заключительные замечания	340
7.6. Упражнения	341
7.7. Онлайн-ресурсы	342
Глава 8. Анализ GitHub: особенности сотрудничества при разработке ПО, графы интересов и многое другое	344
8.1. Обзор	345
8.2. GitHub API	346
8.2.1. Подключение к GitHub API	348
8.2.2. Выполнение запросов к GitHub API	352
8.3. Моделирование данных с помощью графов свойств	355
8.4. Анализ графов интересов в GitHub	359
8.4.1. Начало создания графа интересов	359
8.4.2. Вычисление мер центральности графа	364
8.4.3. Расширение графа интересов ребрами «следования» между пользователями	367
8.4.4. Использование узлов в качестве точек опоры для увеличения эффективности запросов	380
8.4.5. Визуализация графа интересов	387
8.5. Заключительные замечания	388
8.6. Упражнения	390
8.7. Онлайн-ресурсы	391
ЧАСТЬ II. СБОРНИК РЕЦЕПТОВ ДЛЯ TWITTER	393
Глава 9. Сборник рецептов для Twitter	394
9.1. Доступ к Twitter API для целей разработки	395
9.1.1. Задача	395
9.1.2. Решение	395
9.1.3. Пояснение	395
9.2. Использование OAuth для доступа к Twitter API в промышленных целях	397
9.2.1. Задача	397
9.2.2. Решение	397
9.2.3. Пояснение	397
9.3. Поиск актуальных тем	402
9.3.1. Задача	402
9.3.2. Решение	402
9.3.3. Пояснение	402

9.4. Поиск твитов	403
9.4.1. Задача	403
9.4.2. Решение	403
9.4.3. Пояснение	403
9.5. Конструирование удобных вызовов функций	405
9.5.1. Задача	405
9.5.2. Решение	405
9.5.3. Пояснение	406
9.6. Запись и чтение текстовых файлов с данными JSON	407
9.6.1. Задача	407
9.6.2. Решение	407
9.6.3. Пояснение	407
9.7. Сохранение данных JSON в MongoDB и доступ к ним	408
9.7.1. Задача	408
9.7.2. Решение.....	408
9.7.3. Пояснение.....	408
9.8. Получение выборки из потока твитов с использованием Streaming API.....	411
9.8.1. Задача	411
9.8.2. Решение	412
9.8.3. Пояснение	412
9.9. Сбор временных последовательностей данных	413
9.9.1. Задача	413
9.9.2. Решение	414
9.9.3. Пояснение	414
9.10. Извлечение сущностей из твитов	415
9.10.1. Задача	415
9.10.2. Решение	416
9.10.3. Пояснение	416
9.11. Поиск самых популярных твитов в коллекции	417
9.11.1. Задача	417
9.11.2. Решение	417
9.11.3. Пояснение	418
9.12. Поиск самых популярных сущностей в коллекции твитов	419
9.12.1. Задача	419
9.12.2. Решение	419
9.12.3. Пояснение	419

9.13. Вывод результатов частотного анализа в табличной форме.....	420
9.13.1. Задача	420
9.13.2. Решение	421
9.13.3. Пояснение	421
9.14. Поиск пользователей, ретвитнувших статус.....	422
9.14.1. Задача	422
9.14.2. Решение.....	422
9.14.3. Пояснение	422
9.15. Определение автора твита.....	425
9.15.1. Задача	425
9.15.2. Решение	425
9.15.3. Пояснение	425
9.16. Выполнение надежных запросов к Twitter	426
9.16.1. Задача	426
9.16.2. Решение	426
9.16.3. Пояснение	427
9.17. Получение информации из профиля пользователя.....	429
9.17.1. Задача	429
9.17.2. Решение.....	429
9.17.3. Пояснение.....	429
9.18. Извлечение сущностей твитов из произвольного текста.....	431
9.18.1. Задача	431
9.18.2. Решение	431
9.18.3. Пояснение	431
9.19. Получение всех друзей и последователей пользователя	432
9.19.1. Задача	432
9.19.2. Решение	432
9.19.3. Пояснение	432
9.20. Анализ друзей и последователей пользователя	435
9.20.1. Задача	435
9.20.2. Решение	435
9.20.3. Пояснение	435
9.21. Извлечение твитов пользователя	436
9.21.1. Задача	436
9.21.2. Решение	437
9.21.3. Пояснение	437

9.22. Обход графа дружбы	439
9.22.1. Задача	439
9.22.2. Решение	439
9.22.3. Пояснение	439
9.23. Анализ содержимого твитов.....	441
9.23.1. Задача	441
9.23.2. Решение	441
9.23.3. Обсуждение	441
9.24. Обобщение целевых ссылок.....	443
9.24.1. Задача	443
9.24.2. Решение.....	443
9.24.3. Пояснение	443
9.25. Анализ избранных твитов пользователя.....	446
9.25.1. Задача	446
9.25.2. Решение	446
9.25.3. Пояснение	447
9.26. Заключительные замечания	448
9.27. Упражнения	448
9.28. Онлайн-ресурсы.....	450

ЧАСТЬ III. ПРИЛОЖЕНИЯ..... 451

Приложение А. Информация о виртуальной машине с примерами для этой книги.....	452
---	-----

Приложение Б. Основы OAuth.....	454
Обзор	454
OAuth 1.0a.....	455
OAuth 2.0.....	457

Приложение В. Советы и рекомендации для Python и Jupyter Notebook	460
--	-----

Об авторах.....	461
------------------------	-----

Об обложке	462
-------------------------	-----

Если притупится топор, и если лезвие его
не будет отточено, то надобно будет напрягать
силы; мудрость умеет это исправить.

Екклесиаст 10:10

Предисловие

Всемирная паутина — это больше социальное творение, чем техническое.

Я разработал ее не в качестве технической игрушки, а чтобы произвести социальный эффект — помочь людям совместно работать. Конечной целью Всемирной паутины является поддержка и улучшение нашего веб-присутствия в мире. Мы объединяемся в семьи, ассоциации и компании. Мы укрепляем доверие за милю от нас и недоверие за углом.

*Тим Бернерс-Ли (Tim Berners-Lee),
Weaving the Web («Плетя паутину:
истоки и будущее Всемирной паутины»)*

Примечание Мэтью Рассела

Прошло больше пяти лет, как я внес последние штрихи в рукопись второго издания «Data mining», и с тех пор многое изменилось. Я узнал много нового, технологии продолжали развиваться стремительными темпами и сами социальные сети созрели до такой степени, что правительства в настоящее время формулируют правовую основу в отношении сбора, распространения и использования данных.

Понимая, что моя занятость не позволит мне целиком отдалиться работе, чтобы подготовить новое издание книги, освежить и дополнить ее содержимое, но искренне полагая, что пришло время для этого, я осознал необходимость найти соавтора, который поможет обновить книгу для очередной волны предпринимателей, специалистов и хакеров, которым интересна тема анализа социальных сетей. Мне потребовалось больше года, чтобы найти соавтора, который испытывал бы ту же страсть к предмету и обладал навыками и решимостью, необходимыми для работы над книгой.

Я даже не могу выразить, как я благодарен Михаилу Классену (Mikhail Klassen) за его невероятный вклад в сохранение этого плода моих трудов на долгие

годы вперед. Читая книгу, вы сами увидите, какой огромный труд он проделал, обновив код примеров, улучшив доступность среды выполнения и расширив содержимое новой важной главой, — и все это помимо правки и обновления остальной рукописи в целом — и с энтузиазмом передал обновленные знания следующей волне предпринимателей, специалистов и хакеров, интересующихся анализом социальных сетей.

README.1st

Эта книга создавалась, чтобы передать знания конкретной целевой аудитории. И чтобы избежать любой путаницы в отношении ее целей и задач, а также гневных электронных писем и негативных отзывов недовольных читателей или других недоразумений, которые могут возникнуть, в остальной части этого предисловия мы попытаемся помочь вам определить, являетесь ли вы частью этой аудитории. Будучи людьми занятыми, мы высоко ценим свое время и хотим сразу же заверить вас, что считаем это справедливым и для вас. Пусть нам это не всегда удастся, но в реальной жизни мы действительно стараемся уважать своих собеседников больше, чем самих себя, и это предисловие — наша попытка отдать дань уважения вам, дорогой читатель, и дать понять, оправдает ли эта книга ваши ожидания.

Предвосхищая ожидания

В первую очередь эта книга предполагает, что ее читатель хочет узнать, как добывать данные из популярных социальных веб-сетей, избежать технических проблем при опробовании примеров кода и попутно получить *много* удовольствия. Кто-то из вас мог взять эту книгу исключительно с целью познакомиться с кругом возможностей, но спешим вас уверить, что она написана так, чтобы вы действительно могли попрактиковаться на предлагаемых упражнениях и стать специалистами по поиску и извлечению данных, выполнив несколько простых шагов по настройке среды разработки. Если у вас уже есть опыт программирования, вы относительно безболезненно сможете приступить к опробованию примеров кода. Но даже если вы никогда не программировали раньше и считаете себя хотя бы немного подкованными технически, я осмелюсь сказать, что сможете использовать эту книгу как отправную точку в потрясающее путешествие, которое расширит ваши горизонты до пределов, которых вы пока даже не представляете.

Чтобы в полной мере насладиться этой книгой и всем, что она может предложить, вас должны интересовать обширные возможности поиска и добычи данных, спрятанных в недрах популярных социальных сетей, таких как Twitter, Facebook, LinkedIn и Instagram, и у вас должно быть достаточно рвения, чтобы установить Docker, использовать его для запуска виртуальной машины и следовать за примерами кода для Jupyter Notebook, фантастического веб-инструмента, используемого в каждой главе. Обычно для запуска примеров достаточно нескольких нажатий клавиш, так как весь код представлен в удобном пользовательском интерфейсе.

Эта книга даст вам новые знания, которые наверняка порадуют вас, и добавит в ваш инструментарий несколько незаменимых инструментов, но, что еще более важно, она поведает увлекательную историю и не позволит вам скучать на этом пути. Это история о науке, изучающей данные из социальных сетей, о данных, спрятанных внутри них, и некоторых интригующих возможностях, которые открываются перед вами.

Читая эту книгу от начала до конца, вы обнаружите, что повествование плавно перетекает из одной главы в другую. Даже притом, что все главы следуют общему шаблону, знакомя с определенной социальной сетью, рассказывая, как использовать ее API для извлечения данных, и представляя некоторые приемы анализа этих данных, сложность рассматриваемых тем постепенно увеличивается. Начальные главы много времени уделяют знакомству с основными понятиями, тогда как последующие, основывающиеся на предыдущих, постепенно знакомят с широким спектром инструментов и методов анализа социальных сетей, которые вы сможете использовать в других областях, как ученый, аналитик, дальновидный мыслитель или любопытный читатель.

За последние годы некоторые из самых популярных социальных сетей перестали быть чем-то необычным и превратились в обыденное явление, изменив нашу жизнь в интернете и вне его и позволив технологиям выявить в нас лучшие (а иногда и худшие) черты. Вообще говоря, каждая глава этой книги переплетает социальные сети с методами добычи данных, их анализа и визуализации для получения ответов на следующие вопросы:

- Кто с кем знаком и какими общими чертами обладают люди, объединенные их социальной сетью?
- Как часто конкретные люди общаются друг с другом?
- Какие социальные сети наиболее ценятся в конкретной группе людей?
- Как география влияет на наши социальные связи в онлайн-мире?

- Кто в социальной сети пользуется наибольшей популярностью или влиянием?
- О чем рассказывают люди друг другу (и насколько это ценно)?
- Что интересует людей, судя по языку, который они используют в цифровом мире?

Ответы на эти вопросы часто дают ценную информацию и открывают новые возможности (иногда выгодные) для предпринимателей, социологов и других практиков, которые пытаются понять суть задачи и найти решения. Такие задачи, как создание высококлассного приложения с нуля, способного ответить на эти вопросы и выходящего далеко за рамки типичного использования библиотек визуализации, и вообще конструирование чего-либо суперсовременного, не входят в область интересов этой книги. Вы будете очень разочарованы, если купите эту книгу, надеясь с ее помощью сделать что-то подобное. Тем не менее книга описывает основные строительные блоки, которые помогут найти ответы на эти вопросы и дадут точку опоры для создания подобных высококлассных приложений или проведения таких исследований. Просмотрите несколько глав и убедитесь сами. Эта книга много внимания уделяет основам.

Важно отметить, что API социальных сетей постоянно меняются. Социальные сети появились не так давно, и даже те из них, которые сегодня кажутся наиболее устоявшимися, продолжают адаптироваться к особенностям их использования людьми и пытаются противостоять вновь возникающим угрозам безопасности и конфиденциальности. Поэтому их программные интерфейсы (API) тоже могут изменяться, в результате чего примеры кода, приведенные в этой книге, в будущем могут не работать или работать не так, как предполагалось. Мы попытались создать и представить вашему вниманию максимально реалистичные примеры, полезные для общих целей и разработчиков приложений, поэтому некоторые из них требуют отправки заявки на рассмотрение и утверждение¹. Мы постарались снабдить их соответствующими примечаниями, но имейте в виду, что условия использования API могут измениться в любое время. Тем не менее пока ваше приложение соблюдает условия пользования услугами, оно, скорее всего, будет одобрено, так что это стоит усилий.

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

Технологии на основе Python

В этой книге все примеры намеренно написаны на языке программирования Python. Простой и понятный синтаксис Python, удивительная экосистема пакетов, упрощающих доступ к API и управление данными, а также основные структуры данных, практически прямо соответствующие своим аналогам в формате JSON, делают его отличным и мощным инструментом обучения, который легко получить и использовать. Вдобавок к языку Python, который уже сам по себе является отличным педагогичным и прагматичным выбором для анализа социальных сетей, существует еще Jupyter Notebook (<https://jupyter.org/>) — мощный интерактивный интерпретатор кода, который реализует пользовательский интерфейс, напоминающий блокнот, в окне веб-браузера и сочетает возможности выполнения кода, вывода текста и математических формул, графики и многого другого. Трудно представить лучший инструмент для обучения, потому что он существенно упрощает задачу доставки примеров кода, которые вы, как читатель, легко сможете исследовать и выполнять. На рис. П.1 показано, как выглядит интерфейс Jupyter Notebook, со списком примеров для всех глав книги. На рис. П.2 показано, как выглядит один из блокнотов.

Для каждой главы в этой книге имеется соответствующий блокнот Jupyter Notebook с примерами кода, который вы сможете изучить, поэксперименти-



Рис. П.1. Внешний вид списка блокнотов с примерами в интерфейсе Jupyter Notebook

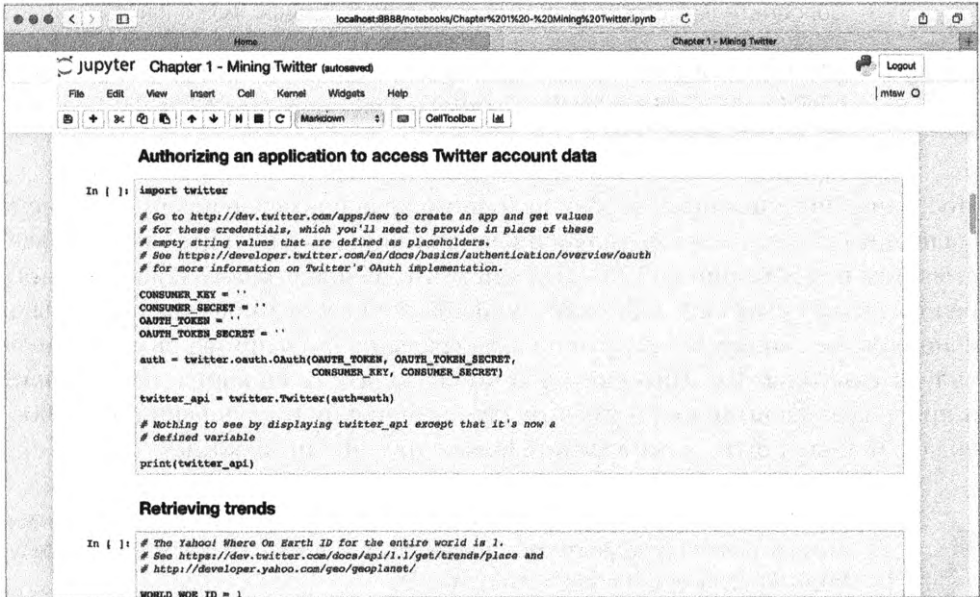


Рис. П.2. Блокнот с примерами для главы 1 — Анализ Twitter

ровать и настроить для своих целей. Имеющие опыт программирования, но не знакомые с синтаксисом Python, пролистав несколько страниц, быстро освоят его, мы верим в это. Желающие без труда найдут в интернете отличную документацию, а если вам нужно исчерпывающее введение в язык программирования Python, обратитесь к официальному учебнику¹. Исходный код примеров на Python в этом издании был переработан и теперь совместим с Python 3.6.

Jupyter Notebook — отличный инструмент, но если вы новичок в мире программирования на Python, с нашей стороны было бы неправильно (и, возможно, немного грубо) посоветовать отыскать инструкции по настройке среды разработки в интернете. Чтобы сделать ваше знакомство с этой книгой как можно более приятным, мы реализовали виртуальную машину под ключ с Jupyter Notebook и всеми другими зависимостями, необходимыми для опробования примеров из этой книги. Вам нужно лишь выполнить несколько простых шагов, и через 15 минут вы будете готовы к экспериментам. Имеющие опыт программирования могут настроить свою среду разработки, но мы надеемся убедить вас, что использование виртуальной машины является лучшей отправной точкой.

¹ <http://bit.ly/1a1kDj8>.



Дополнительные сведения о работе с виртуальной машиной вы найдете в приложении А. Также загляните в приложение В: в нем даются некоторые советы, касающиеся Jupyter Notebook, и описываются распространенные идиомы программирования на Python, широко используемые во всех примерах исходного кода в этой книге.

Последние исправленные версии исходного кода примеров и сопутствующие сценарии для создания виртуальной машины доступны¹ на сайте GitHub, в общественном репозитории Git². Мы надеемся, что наличие такого общественного репозитория будет способствовать укреплению сотрудничества единомышленников, желающих вместе работать над расширением примеров и решением увлекательных задач. Надеемся, что вы создадите свою копию репозитория и приложите свои знания и умения для расширения и улучшения исходного кода — и, может быть, даже заведете новых друзей или знакомых.



Официальный репозиторий GitHub с последними улучшенными и исправленными версиями исходного кода примеров для этой книги доступен по адресу: <http://bit.ly/Mining-the-Social-Web-3E>.

Новое в третьем издании

Как отмечалось выше, к работе над этим, третьим, изданием в качестве соавтора был привлечен Михаил Классен (Mikhail Klassen).

Технологии развиваются очень быстро, и вместе с ними развиваются социальные медиаплатформы. Приступив к пересмотру второго издания, мы быстро поняли, что книга только выиграет, если отразить в ней произошедшие изменения. Первым и наиболее существенным изменением стал перенос кода с Python 2.7 на более позднюю версию Python 3.0+. В мире еще остаются несгибаемые сторонники Python 2.7, тем не менее переход на Python 3 дает много преимуществ, не последним из которых является улучшенная поддержка Юникода. При работе с данными из социальных сетей, которые часто включают эмодзи³ и буквы из других алфавитов, отличных от латинского, очень важно иметь хорошую поддержку Юникода.

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² <http://bit.ly/16mhOep>.

³ Символы, обозначающие эмоции, которые также называют смайликами или эмодзи. — *Примеч. пер.*

В условиях растущей обеспокоенности по поводу конфиденциальности персональных данных платформы социальных сетей совершенствуют свои программные интерфейсы (API), стремясь надежнее защитить пользовательскую информацию, ограничивая доступ к ней сторонним приложениям, даже проверенным и одобренным.

Некоторые примеры кода в предыдущих изданиях этой книги перестали работать из-за этих ограничений. Такие примеры мы заменили новыми, учитывающими ужесточившиеся ограничения, но при этом они все еще иллюстрируют немало интересного.

Некоторые примеры утратили работоспособность из-за существенного изменения API социальных сетей, однако те же самые данные остались доступны, просто извлекать их нужно по-другому. Потратив время на чтение документации разработчика для каждой платформы, мы переделали примеры кода из второго издания, используя в них новые вызовы API.

Но самым большим изменением в третьем издании стала новая глава о работе с социальной сетью Instagram (глава 3). Instagram — чрезвычайно популярная платформа, которую мы не могли обойти стороной. Эта глава дала нам возможность продемонстрировать технологии (в том числе методы глубокого обучения), которые могут пригодиться для анализа изображений. Рассмотрение этой темы может быстро перейти в сугубо техническую плоскость, поэтому мы постарались описать ее максимально доступным способом и использовать мощный API распознавания образов, делающий всю тяжелую работу за нас. В результате у нас получилось несколькими строками на Python реализовать систему, способную проанализировать фотографии, опубликованные в Instagram, и рассказать, что на них изображено.

Еще одним существенным изменением стала глубокая переделка главы 5. Теперь в ней рассказывается об извлечении текстовых файлов без привязки к социальной сети Google+. Основа этой главы осталась прежней, но содержимое более обобщено для работы с любыми API, возвращающими данные на естественном языке.

Попутно было реализовано несколько других технических решений, с которыми отдельные читатели могут не согласиться. В главе об анализе почтовых ящиков (глава 7) во втором издании использовалась MongoDB — база данных для хранения и создания запросов к электронной почте. Системы этого типа играют важную роль, но, решив опробовать код из этой книги вне контейнера Docker, вам придется приложить дополнительные усилия, чтобы установить систему

баз данных. Кроме того, мы хотели показать больше примеров, использующих библиотеку *pandas*, представленную в главе 2. Эта библиотека быстро стала одним из важных инструментов в арсенале специалистов по анализу данных, позволяя манипулировать табличными данными. Было бы неправильно обойти ее стороной в книге, посвященной добыче данных. Тем не менее мы сохранили примеры с MongoDB в главе 9, и, если вы будете использовать контейнер Docker для экспериментов с примерами в этой книге, вы легко сможете опробовать их.

Наконец, мы удалили то, что раньше было главой 9 («Анализ семантической паутины»). Первоначально эта глава была включена в первое издание в 2010 году, однако спустя почти десять лет ее полезность, учитывая направление, в котором продолжили развиваться социальные сети, оказалась сомнительной.



Мы всегда рады конструктивной обратной связи и с удовольствием прочтем ваши отзывы об этой книге, твиты в @SocialWebMining¹ или комментарии на стене в Facebook². Официальный сайт книги и блог с дополнительной информацией можно найти по адресу: <http://MiningTheSocialWeb.com>.

Этические аспекты добычи данных

На момент написания этой книги в Европейском союзе (ЕС) вступил в силу «Общий регламент по защите данных» (General Data Protection Regulation, GDPR). Он определяет, как компании должны защищать конфиденциальность граждан и резидентов ЕС, предоставляя пользователям более полный контроль над их данными. Поскольку многие компании ведут бизнес в Европе, практически все они были вынуждены под угрозой штрафов внести изменения в свои документы, определяющие условия использования и политику конфиденциальности. Регламент GDPR устанавливает новый глобальный базовый уровень конфиденциальности, который, как мы надеемся, положительно повлияет на компании во всем мире, даже не ведущие бизнес в Европе.

Работая над третьим изданием книги «Data mining», мы больше внимания уделили этике использования данных и конфиденциальности пользователей. Во всем мире брокеры данных собирают, сопоставляют и перепродают данные о пользователях интернета: их потребительском поведении, предпочтениях, политических пристрастиях, почтовых индексах, доходах, возрасте и т. д.

¹ <http://bit.ly/1a1kHHzq>.

² <http://on.fb.me/1a1kHPQ>.

В некоторых странах эта деятельность является вполне законной. Обладание достаточным количеством таких данных открывает возможность манипулировать поведением путем целенаправленных сообщений, дизайна интерфейса или публикации сведений, вводящих в заблуждение.

Мы, как авторы книги об анализе данных из социальных сетей и интернета, получающие удовольствие от своей работы, полностью осознаем иронию ситуации. Мы также понимаем, что законные деяния не всегда являются этичными. Добыча данных — это всего лишь набор методов использования определенных технологий, которые сами по себе являются нейтральными в отношении этики. Добытые данные можно использовать с большой пользой. Примером, к которому я (Михаил Классен) часто обращаюсь, может служить UN Global Pulse, инициатива ООН по использованию больших данных (Big Data) для всеобщего блага. Например, используя данные из социальных сетей, можно оценить отношение к инициативам в области здравоохранения (таким, как кампании по вакцинации) или к политическим процессам в стране. Анализируя данные из Twitter, можно быстрее реагировать на возникающие кризисы, такие как эпидемии или стихийные бедствия.

Впрочем, примеры можно найти не только в гуманитарной сфере. Анализ данных широко используется для разработки персонализированных технологий обучения в области образования, а также в некоторых начинающих коммерческих проектах. В других областях анализ данных применяется для прогнозирования пандемий, разработки новых лекарств, определения генов, ответственных за конкретные заболевания, или для проведения профилактического обслуживания двигателя. Ответственно используя данные и уважая конфиденциальность пользователей, можно применять приемы анализа данных, оставаясь в рамках этики, и при этом получать прибыль и создавать нечто новое и потрясающее.

В настоящее время относительно небольшое количество технологических компаний обладают невероятным объемом данных о жизни людей. Они подвергаются растущему давлению со стороны общества и государственного регулирования, вынуждающему их ответственно использовать эти данные. К их чести, многие совершенствуют свои правила и свои API. Прочитав эту книгу, вы лучше узнаете, какие данные может получить сторонний разработчик (например, вы сами) из этих платформ, и познакомитесь со многими инструментами, используемыми для преобразования данных в знания. Мы надеемся, что вы также получите более глубокое представление о злоупотреблениях технологиями. И тогда, как информированный гражданин, вы сможете выступать за разумные законы и защищать частную жизнь каждого.

Типографские соглашения

В этой книге *широко* используются гиперссылки, что делает ее идеальной для чтения в электронном формате, таком как PDF. Многие гиперссылки были сокращены с помощью сервис-службы bit.ly в интересах читателей, купивших печатное издание. Все гиперссылки проверены.

В этой книге приняты следующие типографские соглашения:

Курсив

Используется для обозначения новых терминов.

Моноширинный шрифт

Применяется для оформления листингов программ и программных элементов в обычном тексте, таких как имена переменных и функций, баз данных, типов данных, переменных окружения, инструкций и ключевых слов.

Моноширинный жирный

Обозначает команды или другой текст, который должен вводиться пользователем. Также иногда используется в листингах для привлечения внимания.



Так обозначаются примечания общего характера.



Так выделяются советы и предложения.



Так обозначаются предупреждения и предостережения.

Использование примеров программного кода

Последние версии исходного кода примеров для этой книги хранятся на сайте GitHub по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Это официальный репозиторий примеров для книги. Мы рекомендуем периодически заглядывать в него и проверять появление последних исправлений и дополнений, выполненных авторами и другими членами сообщества. Если вы читаете печатную версию книги, есть вероятность, что примеры кода в ней могут оказаться устаревшими, но пользуясь репозиторием GitHub, вы всегда будете иметь последние исправленные версии примеров. Решив использовать виртуальную машину для опробования примеров, как предлагается в этой книге, вы всегда будете иметь последнюю версию исходного кода, а если вы решите настроить свою среду разработки, не забудьте загрузить архив с исходным кодом непосредственно из репозитория GitHub.



Пожалуйста, сообщайте о проблемах с примерами непосредственно в репозитории GitHub, а не в каталоге O'Reilly. По мере устранения проблем в исходном коде на GitHub обновления переносятся обратно в рукопись книги, которая затем периодически рассылается читателям в качестве обновления электронной версии.

В общем случае все примеры кода из этой книги вы можете использовать в своих программах и в документации. Вам не нужно обращаться в издательство за разрешением, если вы не собираетесь воспроизводить существенные части программного кода. Например, если вы разрабатываете программу и используете в ней несколько отрывков программного кода из книги, вам не нужно обращаться за разрешением. Однако в случае продажи или распространения компакт-дисков с примерами из этой книги вам следует получить разрешение от издательства O'Reilly. Если вы отвечаете на вопросы, цитируя данную книгу или примеры из нее, получение разрешения не требуется. Но при включении существенных объемов программного кода примеров из этой книги в вашу документацию вам необходимо будет получить разрешение издательства.

За получением разрешения на использование значительных объемов программного кода примеров из этой книги обращайтесь по адресу permissions@oreilly.com.

Благодарности к третьему изданию

Я (Михаил Классен) не принял бы участия в работе над этой книгой, если бы не случайная встреча со Сюзан Конант (Susan Conant) из O'Reilly Media. Она увидела потенциал для сотрудничества с Мэтью Расселом в работе над третьим изданием этой книги, и я был рад принять участие в этом проекте. Мне посчастливилось работать с великолепной редакционной командой издательства O'Reilly, и я хотел бы поблагодарить Тима Макговерна (Tim McGovern), Элли Макдональд (Ally MacDonald) и Алисию Янг (Alicia Young). Вместе с этой книгой в O'Reilly была подготовлена серия видеолекций, и я хочу также поблагодарить команду, работавшую над ними вместе со мной: Дэвида Кейтса (David Cates), Питера Онга (Peter Ong), Адама Ритца (Adam Ritz) и Аманду Портер (Amanda Porter).

Работать над проектом только по вечерам и выходным — значит отнимать время у семьи, поэтому спасибо моей жене Шейле за понимание.

Благодарности ко второму изданию

Я (Мэтью Рассел) в благодарностях к первому изданию писал, что работа над книгой — огромная жертва. Время, которое вы проводите в отрыве от друзей и семьи (что происходит в течение длительного периода по ночам и выходным), обходится очень дорого, его не вернуть, и вам действительно нужна определенная моральная поддержка, чтобы преодолеть этот сложный период без ущерба для отношений. Еще раз спасибо моим друзьям и семье за то, что терпели, пока я писал еще одну книгу, и, вероятно, думали, что у меня есть какое-то хроническое расстройство, вынуждающее меня работать по ночам и выходным. Если вы найдете клинику, занимающуюся реабилитацией людей, зависимых от написания книг, я обещаю, что пойду и проверю себя.

Любому проекту нужен хороший руководитель, и мне посчастливилось работать с удивительным редактором Мэри Трезелер (Mary Treseler) и ее потрясающими коллегами. Работа над технической книгой — это, мягко говоря, долгий и тяжелый труд, и я получил замечательный опыт работы с профессионалами, готовыми помочь вам пройти через этот утомительный процесс и создать отполированный продукт, который можно с гордостью представить миру. Кристен Браун (Kristen Brown), Рейчел Монаган (Rachel Monaghan) и Рейчел Хед (Rachel Head) сделали все возможное, чтобы вывести меня на совершенно новый уровень профессионализма.

Подробные комментарии, которые я получал от моих одаренных редакторов и научных редакторов, тоже нельзя охарактеризовать иначе как восхитительные. Обратная связь, начиная от сугубо технических рекомендаций и заканчивая советами по использованию лучших практик программирования на Python и как лучше преподнести материал целевой аудитории, превзошла все мои ожидания. Книга, которую вы держите в руках, не получилась бы такой качественной без полученных мной ценных советов и рекомендаций. Большое спасибо вам, Эйб Мьюзик (Abe Music), Николас Мэйн (Nicholas Mayne), Роберт П. Дж. Дэй (Robert P.J. Day), Рэм Нарасимхан (Ram Narasimhan), Джейсон Йи (Jason Yee) и Кевин Макис (Kevin Makice), за ваши очень подробные отзывы. Они оказали огромное влияние на качество этой книги, и я сожалею только о том, что у нас не было возможности сотрудничать более тесно. Спасибо также Тейту Эскею (Tate Eskew) за то, что познакомил меня с Vagrant, инструментом, кардинально упростившим создание простой и удобной виртуальной машины для этой книги.

Хочу также поблагодарить моих коллег из Digital Reasoning за содержательные беседы об анализе данных и на другие компьютерные темы, которые мы вели на протяжении многих лет, и помогли мне сформировать мое профессиональное мышление. Это счастье быть частью такой команды талантливых и одаренных людей. Отдельное спасибо Тиму Эстесу (Tim Estes) и Робу Меткалфу (Rob Metcalf), поддерживавшим меня в работе над трудоемкими проектами (вне моих прямых обязанностей в Digital Reasoning), такими как написание книг.

И наконец, спасибо всем, кто читал и адаптировал исходный код примеров для этой книги и делился конструктивными замечаниями на протяжении всего срока, пока первое издание оставалось актуальным. Вас очень много, чтобы я мог назвать вас всех, и вы все оказали огромную помощь в подготовке этого, второго, издания. Я надеюсь, что это издание оправдывает ваши ожидания и окажется в вашем списке полезных книг, которые вы порекомендуете друзьям или коллегам.

Благодарности к первому изданию

Работа над книгой, особенно над технической книгой, требует огромных жертв. Дома я уделяю много внимания жене Базирет (Baseeret) и дочери Линдси Белль (Lindsay Belle), чем очень горжусь. Больше спасибо вам обоим, что любите меня, несмотря на мои амбиции когда-нибудь захватить мир. (Это просто один из этапов на пути к цели, и я стараюсь преодолеть его — честно.)

Я искренне верю, что сумма ваших решений привела вас туда, где вы находитесь (особенно в профессиональной карьере), но никто и никогда не сможет завершить путешествие в одиночку, и для меня большая честь отдать должное, когда наступает срок. Я действительно рад, что был окружен одними из самых ярких людей в мире, когда работал над этой книгой, включая технического редактора Майка Лоукидеса (Mike Loukides), талантливых сотрудников из O'Reilly, ревностных рецензентов и всех тех, кто помог мне завершить эту книгу. Особенно я хочу поблагодарить Эйба Мьюзика (Abe Music), Пита Уордена (Pete Warden), Тантека Челика (Tantek Celik), Дж. Криса Андерсона (J. Chris Anderson), Сальваторе Санфилиппо (Salvatore Sanfilippo), Роберта Ньюсона (Robert Newson), Ди Джея Патила (DJ Patil), Чимези Огбуджа (Chimezie Ogbuji), Тима Голдена (Tim Golden), Брайана Кертина (Brian Curtin), Раффи Крикориана (Raffi Krikorian), Джеффа Хаммербахера (Jeff Hammerbacher), Ника Дукоффа (Nick Ducoff) и Кэмерона Марлоу (Cameron Marlowe) за рецензирование рукописи и множество полезных комментариев, которые, без сомнения, помогли достичь лучшего результата. Также хочу поблагодарить Тима О'Рейли (Tim O'Reilly) за то, что любезно позволил мне поместить некоторые из его данных из Twitter и Google+ под микроскоп; это определенно сделало соответствующие главы гораздо более интересными. К сожалению, я не могу перечислить поименно всех, кто прямо или косвенно повлиял на мою жизнь или на эту книгу.

Наконец, спасибо вам, дорогой читатель, что дали шанс этой книге. Если вы читаете эти строки, значит, как минимум, задумываетесь о ее покупке. И если вы сделаете это, возможно вы найдете в ней какие-то недочеты, несмотря на все мои усилия; тем не менее я верю, что вы приятно проведете несколько вечеров/выходных, и вам посчастливится научиться чему-нибудь новому.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

Часть I

ЭКСКУРСИЯ ПО СОЦИАЛЬНЫМ СЕТЯМ

В первой части книги представлены некоторые практические приемы получения немедленной отдачи от некоторых из самых популярных социальных сетей. Здесь вы узнаете, как получить доступ к API для анализа социальных данных из Twitter, Facebook, LinkedIn, Instagram, веб-страниц, блогов и лент новостей, электронных писем и учетных записей GitHub. Каждая глава в этой части стоит особняком и рассказывает свою историю, но все вместе они рассказывают более широкую общую историю. Сложность каждой следующей главы постепенно увеличивается, при этом некоторые методы или технологии, представленные в первых главах, находят повторное применение в последующих.

Вследствие такого постепенного увеличения сложности я советую читать главы последовательно, но вы можете также выбирать главы по своему усмотрению и исследовать примеры, приводящиеся в них. Примеры кода для каждой главы объединены в один блокнот Jupyter Notebook, названный в соответствии с номером главы в этой книге.



Исходный код примеров доступен на GitHub¹. Кроме того, мы настоятельно рекомендуем воспользоваться возможностями Docker для создания виртуальной машины. Это позволит вам работать с примерами кода в предварительно настроенной среде разработки, которая «просто работает».

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

Вступление

Я уже говорил об этом в предисловии и буду повторять в каждой главе, что это не типичная техническая книга с архивом примеров кода, сопровождающего текст. Эта книга пытается изменить статус-кво и определить новый стандарт технической литературы, в которой примеры являются самым настоящим проектом с открытым исходным кодом, причем сама книга представляется как форма поддержки класса «премиум» этой кодовой базы.

Для достижения этой цели были приложены все силы, чтобы объединить обсуждение в книге с примерами кода в максимально гладкий процесс обучения. После долгих обсуждений с читателями первого издания и размышлений над извлеченными уроками стало очевидно, что лучшим решением является интерактивный пользовательский интерфейс, поддерживаемый сервером в виртуальной машине и снабженный надежной конфигурацией. Не существует более простого и удачного способа дать вам полный контроль над кодом и гарантировать, что код будет «просто работать», независимо от используемой операционной системы — будь то macOS, Windows или Linux — и от изменений в API используемого стороннего программного обеспечения, которые могут нарушить совместимость.

Для создания виртуальной машины в третьем издании книги была использована вся мощь Docker. Docker — это технология, поддержку которой можно установить в любой из распространенных операционных систем и использовать для создания «контейнеров» и управления ими. Контейнеры Docker действуют подобно виртуальным машинам, создавая автономную среду со всем необходимым исходным кодом, выполняемыми файлами и зависимостями, необходимыми для запуска этого программного обеспечения. Для многих сложных программных продуктов существуют контейнерные версии, что делает их установку простой и легкой в любой системе, где может работать Docker.

В репозитории GitHub¹ этой книги теперь имеется файл `Dockerfile`. Такие файлы подобны рецептам, которые инструктируют Docker, как «собрать» программное обеспечение в контейнер. Подробнее о том, как быстро установить и запустить контейнер, рассказывается в приложении А.

Воспользуйтесь преимуществами этого мощного инструмента для обучения в интерактивном режиме.



Инструкции по подготовке виртуальной машины для опробования примеров из второго издания можно найти в книге «Reflections on Authoring a Minimum Viable Book»².

Хотя логичнее вслед за этим вступлением перейти сразу к главе 1, я все же посоветовал бы воспользоваться моментом и прочитать приложения А и В, где рассказывается, как подготовиться к опробованию примеров кода. Приложение А ссылается на онлайн-документ и сопровождающие его скринкасты, которые помогут вам легко и быстро настроить Docker для создания виртуальной машины с примерами из этой книги. Приложение В тоже ссылается на онлайн-документ со справочной информацией, которая может пригодиться для организации максимально эффективной работы с виртуальной машиной.

Даже если вы опытный разработчик, способный проделать всю эту работу самостоятельно, все равно попробуйте воспользоваться Docker для исследования примеров из этой книги, чтобы избавить себя от неизбежных сложностей, связанных с установкой программного обеспечения.

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² <http://bit.ly/1a1kPyJ>.

1

Twitter: исследование актуальных тем, о чем говорят люди и многое другое

Поскольку это первая глава, не будем торопиться и не спеша начнем наше путешествие по миру анализа социальных сетей. Учитывая, что данные в Twitter настолько доступны и открыты для общественности, в главе 9 будет представлен широкий спектр возможностей анализа данных в виде краткой коллекции рецептов в удобном формате «задача/решение», которые легко можно адаптировать и применить к широкому кругу проблем. Кроме того, для исследования данных из Twitter вы сможете использовать приемы, описываемые в последующих главах.



Загрузите последний исправленный исходный код примеров для этой (и любой другой) главы с сайта GitHub¹. Воспользуйтесь также преимуществами виртуальной машины, описанными в приложении А, чтобы получить максимальное удовольствие от опробования примеров кода.

1.1. Обзор

В этой главе мы познакомимся с минимальной (но эффективной) средой разработки на языке Python, рассмотрим программный интерфейс Twitter и сделаем некоторые аналитические выводы на основе твитов с применением методов частотного анализа. В этой главе рассматриваются следующие темы:

- платформа разработчиков Twitter и отправка запросов программному интерфейсу;
- метаданные твитов и их использование;

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

- извлечение из твитов таких сведений, как упоминания пользователей, хештеги и адреса URL;
- приемы и методы реализации частотного анализа на Python;
- построение гистограмм в Jupyter Notebook на основе данных из Twitter.

1.2. Причины популярности Twitter

В большинстве глав вы не найдете глубокомысленных рассуждений, но поскольку это первая глава книги и она знакомит с социальной сетью, суть которой часто понимают неправильно, представляется целесообразным воспользоваться моментом, чтобы познакомиться с Twitter на фундаментальном уровне.

Как бы вы описали Twitter?

На этот вопрос можно ответить по-разному, но давайте рассмотрим его с точки зрения, затрагивающей некоторые фундаментальные общечеловеческие аспекты, которые должна учитывать любая технология, чтобы быть полезной и успешной. В конце концов, цель технологий заключается в улучшении нашего человеческого опыта.

Что мы как люди хотим от технологий?

- Мы хотим, чтобы нас услышали.
- Мы хотим удовлетворить свое любопытство.
- Мы хотим, чтобы ею легко было пользоваться.
- Мы хотим получить ее прямо сейчас.

В контексте текущей дискуссии это лишь несколько наблюдений, которые в целом справедливы для всего человечества. У нас есть глубоко укоренившаяся потребность делиться своими идеями и опытом, что дает нам возможность общаться с другими людьми, быть услышанными и почувствовать свою ценность и важность. Нам интересно, как устроен окружающий мир, как им управлять, и мы используем общение, чтобы делиться своими наблюдениями, задавать вопросы и вступать в содержательное обсуждение наших проблем с другими людьми.

Два последних пункта подчеркивают присущую нам нетерпимость к затруднениям. В идеале, мы не хотим усердствовать больше, чем это необходимо для удовлетворения нашего любопытства или выполнения какой-то работы;

мы предпочли бы заняться «чем-то еще» или пойти дальше, потому что наше время на этой планете так драгоценно и его так мало. Точно так же мы хотим, чтобы все было прямо *сейчас*, и проявляем нетерпение, когда движение вперед отстает от скорости нашей мысли.

Описать Twitter можно как службу микроблогов, позволяющую людям обмениваться путем изменения строки статуса в соответствии со своими мыслями или идеями. Первоначально размер твитов был ограничен 140 символами, и, хотя этот предел был увеличен, он снова может измениться в будущем. С этой точки зрения Twitter можно рассматривать как бесплатную высокоскоростную глобальную службу обмена текстовыми сообщениями. Иначе говоря, это часть важной инфраструктуры, обеспечивающей быструю и легкую связь. Однако это еще не все. Люди жаждут общения и хотят, чтобы их слышали, и ежемесячно в Twitter выражают идеи, общаются друг с другом напрямую и удовлетворяют свое любопытство 335 миллионов активных пользователей по всему миру¹.

Кроме возможностей макроуровня для маркетинга и рекламы, которые всегда прибыльны с пользовательской базой такого размера, предпосылки для появления такой действительно интересной пользовательской базы создает еще и динамика сети, и в этом заключается такая высокая популярность Twitter. Даже притом, что наличие канала, позволяющего пользователям делиться короткими сообщениями со скоростью мысли, является *необходимым* условием популярности и привлекательности Twitter, одного этого *недостаточно*. Дополнительный ингредиент, который создает достаточное условие, заключается в асимметрии модели следования, помогающей нам удовлетворить свое любопытство. Эта асимметричная модель следования, которая превращает Twitter в нечто более интересное, чем простая социальная сеть, и программный интерфейс обеспечили достаточную основу для кристаллизации самоорганизующегося поведения из хаоса.

Иначе говоря, в отличие от некоторых других социальных сетей, таких как Facebook и LinkedIn, требующих взаимного подтверждения связи между пользователями (что обычно подразумевает реальную связь между ними), модель следования в Twitter позволяет быть в курсе последних событий *любого* другого пользователя, даже если тот не пожелал подписаться на вас или вообще не подозревает о вашем существовании. Модель *подписки* в Twitter проста, но использует фундаментальный аспект, делающий нас людьми: наше любопытство. Twitter дает безграничные возможности для удовлетворения нашего любопытства, будь то увлечение сплетнями о знаменитостях, желание быть в курсе дел

¹ <http://bit.ly/2p2GSV0>.

в любимой спортивной команде, живой интерес к конкретной политической теме или желание общаться с новыми людьми.



В предыдущем абзаце я не случайно определил отношения в Twitter как отношения «следования» (подписки), однако иногда акт *следования* за кем-то описывается как «дружба» (хотя это довольно странная разновидность односторонней дружбы). В официальной документации с описанием Twitter API можно даже столкнуться с определением «друзья»¹, однако лучше все же воспринимать отношения в Twitter как отношения следования, как я и описал.

Представьте способ моделирования связей между людьми и их произвольными интересами в виде *графа интересов*. Графы интересов открывают бесчисленные возможности в области анализа данных, и в первую очередь измерение тесноты связей для выработки интеллектуальных рекомендаций и других целей машинного обучения. Например, используя граф интересов, можно измерить корреляции и давать рекомендации, например: на кого подписаться в Twitter, что покупать в интернете и с кем встречаться. Продолжая аналогию Twitter с графом интересов, имейте в виду, что пользователь Twitter не обязательно должен быть реальным человеком; это вполне может быть неодушевленный объект, компания, музыкальная группа, воображаемая персона, олицетворение кого-то (живого или мертвого) или что-то еще.

Например, @HomerJSimpson² — официальный аккаунт Гомера Симпсона, персонажа сериала «Симпсоны». Гомер Симпсон не является реальным человеком, но хорошо известен во всем мире, и личность @HomerJSimpson в Twitter действует как его канал (или, точнее, его создателей) для привлечения поклонников. Для этой книги тоже имеется официальный аккаунт @SocialWebMining (хотя он едва ли достигнет популярности Гомера Симпсона), предоставляющий читателям книги возможность общения на различных уровнях. Поняв, что Twitter позволяет создавать сообщества интересующихся определенной темой и исследовать их, ценность данных, которые можно извлечь из Twitter, становится более очевидной.

Возможности управления аккаунтами в Twitter ограничиваются лишь добавлением значков, идентифицирующих знаменитостей и общественных деятелей как «верифицированные аккаунты», и положениями соглашения об условиях предоставления услуг Twitter³, которое пользователь должен принять при

¹ <http://bit.ly/2QskIYD>.

² <http://bit.ly/1a1kQD1>.

³ <http://bit.ly/1a1kRXI>.

регистрации. Это тонкое, но важное отличие от некоторых социальных сетей, где аккаунты должны соответствовать реальным, живым людям, предприятиям или организациям, которые вписываются в конкретную таксономию (классификацию). Twitter не накладывает особых ограничений на личность, которой принадлежит аккаунт, и полагается на самоорганизующееся поведение для создания определенного порядка в системе, выражающееся в отношениях следования и фолксономий (folksonomies) — сообществ, возникающих в результате использования хештегов.

ТАКСОНОМИЯ И ФОЛКСОНОМИЯ

Важнейшей особенностью человеческого разума является стремление к классификации и построению иерархий, в которых каждый элемент «принадлежит» родительскому элементу (или является его «потомком»), находящемуся в иерархии уровнем выше. Не вдаваясь в тонкости различий таксономии и онтологии¹, *таксономию* можно рассматривать как иерархическую структуру, например дерево, которая классифицирует элементы в конкретные отношения «родитель/потомок», а *фолксономию*² (folksonomy — термин, придуманный примерно в 2004 году) — как практику совместной классификации и социальной индексации информации в различных экосистемах интернета. Фактически термин «фолксономия» состоит из двух слов: *folk* (народ) и *taxonomy* (таксономия). То есть, по сути, фолксономия — это просто необычный способ описания децентрализованной вселенной тегов, которая возникла как механизм *коллективного разума*, который позволяет людям классифицировать информацию с помощью ярлыков. Одна из замечательных особенностей хештегов в Twitter заключается в том, что органически возникающие фолксономии действуют как точки выражения общих интересов и дают возможность целенаправленного изучения, оставляя при этом возможность для почти неограниченного количества открытий.

1.3. Twitter API

Определив систему координат для Twitter, перейдем теперь к проблеме получения данных из этой социальной сети и их анализа.

1.3.1. Базовая терминология Twitter

Twitter можно охарактеризовать как социальную службу микроблогинга, действующую в режиме реального времени и позволяющую пользователям

¹ <http://bit.ly/1a1kRXy>.

² <http://bit.ly/1a1kU5C>.

публиковать короткие посты — *твиты*, которые появляются в лентах. Твиты могут включать сущности до 280 символов (в настоящее время) текста и ссылки на одно или несколько местоположений в реальном мире. Понимание, как организованы аккаунт, твиты и ленты постов, совершенно необходимо для эффективного использования Twitter API¹, поэтому мы должны кратко познакомиться с этими базовыми понятиями, прежде чем перейдем к взаимодействию с API для извлечения данных. Мы уже говорили о пользователях Twitter и асимметричной модели отношений следования, поэтому в этом разделе кратко рассмотрим твиты и ленты, чтобы сформировать законченное общее понимание платформы Twitter.

Твиты — это суть Twitter, и хотя они условно считаются короткими текстовыми строками, определяющими статус пользователя, в твитах имеется немного больше метаданных, чем может показаться на первый взгляд. Кроме текстового содержимого в твите присутствуют два дополнительных блока метаданных, имеющих особое значение: *сущности* и *местоположения*. Сущности — это упоминания пользователей, хештеги, ссылки URL и медиафайлы, связанные с твитом, а местоположения — это места в реальном мире, которые можно присоединить к твиту. Обратите внимание, что местоположение может определяться географическими координатами, где был создан твит, или это может быть ссылка на место, описанное в твите.

Для большей конкретики рассмотрим пример твита со следующим текстом:

@ptwobrussell is writing @SocialWebMining, 2nd Ed. from his home office in Franklin, TN. Be #social: <http://on.fb.me/16WJAf9>

Этот твит содержит 124 символа текста и 4 сущности: упоминание пользователей @ptwobrussell и @SocialWebMining, хештег #social и URL-адрес <http://on.fb.me/16WJAf9>. Несмотря на то что в тексте присутствует упоминание географического местоположения — Franklin, TN (город Франклин, штат Теннесси), — в метаданных *местоположений* могут быть указаны географические координаты места, где находился пользователь в момент создания твита, не совпадающие с координатами города Франклин в штате Теннесси. Этот довольно большой объем метаданных упакован менее чем в 140 символов, что наглядно показывает, насколько информативным может быть короткое сообщение: оно может однозначно ссылаться на нескольких других пользователей Twitter, на веб-страницы, а также на перекрестные темы с хештегами, которые действуют как

¹ <http://bit.ly/1a1kSKQ>.

точки агрегирования и горизонтально разрезают вселенную Twitter, упрощая поиск в ней.

Наконец, *ленты сообщений* — это хронологически упорядоченные коллекции твитов. В общем случае можно сказать, что лента сообщений — это любая конкретная коллекция твитов, отображаемых в хронологическом порядке; однако чаще отображается пара лент сообщений. С точки зрения произвольного пользователя Twitter, *домашняя лента* — это то, что он видит сразу после входа в аккаунт. В ней отображаются все твиты пользователей, за которыми он следует. Кроме домашней ленты есть также *пользовательская лента*, содержащая твиты, написанные определенным пользователем.

Например, после входа в свой аккаунт Twitter вы будете наблюдать свою домашнюю ленту по адресу <https://twitter.com>. Чтобы увидеть ленту любого конкретного пользователя, в этот URL-адрес нужно добавить идентификатор этого пользователя, например: <https://twitter.com/SocialWebMining>. Если вам интересно узнать, кто следует за конкретным пользователем, вы можете получить список последователей, добавив слово *following* в URL. Например, увидеть ленту сообщений, которую Тим О'Рейли (Tim O'Reilly) наблюдает после входа в Twitter, можно по адресу: <https://twitter.com/timoreilly/following>.

Существует приложение TweetDeck, предлагающее несколько настраиваемых представлений, которые помогают ориентироваться в бурном море твитов, как показано на рис. 1.1. Его стоит попробовать, если прежде вы не выходили далеко за пределы пользовательского интерфейса Twitter.com.

Кроме лент сообщений — коллекций твитов с относительно низкой скоростью обновления — имеются также *потоки*, отображающие течение публичных твитов в реальном времени. Известно, что во время *событий, вызывающих широкий интерес*, таких как дебаты кандидатов на пост президента или крупные спортивные мероприятия, через поток публичных сообщений в пиковые моменты могут протекать сотни тысяч сообщений в минуту¹. Это слишком большой объем данных, чтобы его можно было рассматривать в рамках этой книги, и он порождает интересные инженерные проблемы, что является одной из причин, почему различные сторонние коммерческие поставщики услуг сотрудничают с Twitter, стремясь передать этот поток в массы в виде, более пригодном к использованию. Тем не менее существует ограниченная случайная выборка из ленты публичных сообщений, предлагающая разработчикам API доступ с фильтрацией к достаточному объему публичных данных для создания мощных приложений.

¹ <http://bit.ly/2xenpnR>.



Рис. 1.1. TweetDeck предлагает настраиваемый пользовательский интерфейс, который может пригодиться для анализа происходящего в Twitter и демонстрирует, какого вида данные доступны через Twitter API

В оставшейся части этой главы и во второй части книги будет предполагаться, что у вас есть аккаунт Twitter, необходимый для доступа к Twitter API. Если у вас пока нет аккаунта, найдите время и создайте его, а затем ознакомьтесь с довольно либеральными условиями использования Twitter¹, с описанием API² и правилами разработчика³. Примеры кода в этой главе и во второй части книги не требуют, чтобы у вас были друзья или фолловеры, но некоторые примеры во второй части будут намного интереснее, если у вас есть активный аккаунт с несколькими друзьями и фолловерами, который можно использовать как основу для анализа социальной сети. Если вы не проявляете активности в Twitter, сейчас самое время подключиться и начать заполнять его, чтобы получить больше удовольствия от анализа данных в будущем.

1.3.2. Подключение к Twitter API

Команда Twitter позаботилась о создании простого и удобного в использовании RESTful API. Тем не менее существуют замечательные библиотеки, еще

¹ <http://bit.ly/2e63DVY>.

² <http://bit.ly/1a1kSKQ>.

³ <http://bit.ly/2MsrryS>.

больше упрощающие выполнение запросов к API. Особенно интересен пакет `twitter` для Python, который обертывает Twitter API и имитирует семантику общедоступного API почти «один в один». Как и большинство других пакетов для Python, его можно установить с помощью `pip`, выполнив в терминале команду `pip install twitter`. Если вам не нравится библиотека `twitter`, на выбор есть множество других. Одна из самых популярных альтернатив — `tweepy`.



Инструкции по установке `pip` вы найдете в приложении В.

СОВЕТ: ИСПОЛЬЗУЙТЕ PYDOC ДЛЯ ЭФФЕКТИВНОЙ ПОМОЩИ ПРИ РАЗРАБОТКЕ

Далее мы увидим несколько примеров, иллюстрирующих использование пакета `twitter`, но на всякий случай, если вам понадобится помощь (а она понадобится), стоит запомнить, что всегда можно заглянуть в документацию к пакету (его `pydoc`¹). Вне оболочки Python можно запустить команду `pydoc` в терминале, передав ей имя пакета, который должен находиться в одном из каталогов, перечисленных в `PYTHONPATH`. Например, в Linux или macOS можно просто ввести `pydoc twitter` в терминале и получить документацию с описанием пакета в целом, а если ввести команду `pydoc twitter.Twitter`, она выведет документацию с описанием класса `Twitter`, включенного в этот пакет. В Windows тоже можно получить эту информацию, хотя и немного другим способом: запустив `pydoc` как пакет. Например, команда `python -mpydoc twitter.Twitter` выведет информацию о классе `twitter.Twitter`. Если вам часто приходится обращаться к документации для разных модулей, передайте команде `pydoc` ключ `-w`, и тогда вы получите в ответ HTML-страницу, которую можно сохранить и добавить в закладки в браузере.

Однако чаще помощь нужна прямо в процессе работы над кодом, внутри оболочки Python. В этом случае можно воспользоваться встроенной функцией `help`, которая принимает имя пакета или класса. Пользователи IPython могут ввести имя пакета или класса и добавить знак вопроса, чтобы получить встроенную справку. Например, в обычной оболочке Python можно ввести `help(twitter)` или `help(twitter.Twitter)`, а в IPython или Jupyter Notebook можно использовать более краткую форму `twitter?` или `twitter.Twitter?`.

В качестве стандартной оболочки Python, при работе вне Jupyter Notebook, настоятельно рекомендуется использовать оболочку IPython, потому что она имеет множество удобных функций, таких как автозавершение по клавише табуляции, история сеансов и «магические функции»². Напомним, что в приложении А вы найдете некоторые сведения о рекомендуемых инструментах разработчика, таких как IPython.

¹ <http://bit.ly/1a1kVXg>.

² <http://bit.ly/2nII3ce>.



Для работы с программным интерфейсом Twitter мы выбрали Python, потому что пакет `twitter` очень точно имитирует RESTful API. Если вам интересно узнать, как выполнять запросы на низком уровне, с использованием протокола HTTP, или вы предпочитаете изучать API в более интерактивном режиме, обратитесь к документации разработчика¹, где описывается, как использовать инструмент, такой как `Twurl`, для исследования Twitter API.

Прежде чем вы сможете послать первый запрос к Twitter API, нужно создать приложение на <https://dev.twitter.com/apps>. Для разработчиков это стандартный способ получить доступ к API, а для Twitter — способ контроля и взаимодействия со сторонними разработчиками. Из-за допущенных прежде злоупотреблений платформами социальных сетей теперь требуется подать заявку на аккаунт разработчика Twitter² и получить одобрение создания новых приложений. Вместе с приложением также будет создан набор токенов аутентификации, которые позволят вам получить программный доступ к платформе Twitter.

В данном случае подразумевается, что *вы* создаете приложение, которое собираетесь авторизовать для доступа к данным *своей* учетной записи, поэтому такой подход может показаться немного странным: почему бы просто не использовать свое имя пользователя и пароль для доступа к API? Дело в том, что такой подход может быть неплох для вас, но другие, например ваш друг или коллега, могут чувствовать себя не совсем комфортно, передавая комбинацию имени пользователя и пароля, чтобы получить возможность пользоваться результатами работы вашего приложения. Передача учетных данных — всегда плохая идея. К счастью, эту проблему заметили и создали стандартный протокол открытой авторизации под названием OAuth³ (сокращенно от Open Authorization), который предназначен для подобных ситуаций и в целом для социальных сетей. На данный момент протокол является стандартом для социальных сетей.

Если вы ничего не поняли из вышесказанного, просто запомните, что OAuth дает пользователям возможность разрешить сторонним приложениям получить доступ к данным своей учетной записи, никому не передавая конфиденциальной информации, такой как пароль. Более широкий обзор устройства и осо-

¹ <http://bit.ly/2OpiF6c>.

² <http://bit.ly/2AHBW03>.

³ <http://bit.ly/1a1kZWN>.

бенностей работы протокола OAuth вы найдете в приложении Б. Кроме того, сведения о конкретной реализации можно найти в документации¹ с описанием поддержки протокола OAuth в Twitter.²

Для простоты разработки из параметров вновь созданного приложения следует получить и сохранить такие ключевые элементы информации, как ключ получателя (consumer key), секрет получателя (consumer secret), токен доступа (access token) и секрет токена доступа (access token secret). Вместе эти учетные данные будут использоваться для авторизации пользователя через серию перенаправлений, поэтому относитесь к ним с тем же вниманием, как и к паролям.



Дополнительные сведения об использовании протокола OAuth 2.0, которые понадобятся при создании приложения, требующего авторизации произвольного пользователя для доступа к данным учетной записи, вы найдете в приложении Б.

На рис. 1.2 показана страница, где можно получить эту информацию.

А теперь без дальнейших церемоний создадим аутентифицированное соединение с Twitter API и узнаем, о чем говорят люди, исследовав актуальные темы через ресурс `GET trends/place`³. Между делом добавьте в закладки ссылки на официальную документацию с описанием API⁴, справочник по API⁵, потому что вы регулярно будете обращаться к ним, разбираясь во вселенной Twitter с позиции разработчика.



По состоянию на март 2017 года действовала версия Twitter API 1.1. Кое в чем она существенно отличается от предыдущей версии API v1, с которой вам, возможно, приходилось сталкиваться. Версия API v1 закончила цикл завершения поддержки примерно через шесть месяцев после этого и больше не работает. Все примеры кода в этой книге предполагают работу с версией API 1.1.

¹ <http://bit.ly/2NawA3v>.

² Даже притом, что это особенность конкретной реализации, стоит отметить, что Twitter API v1.1 по-прежнему реализует OAuth 1.0a, тогда как многие другие социальные сети перешли на OAuth 2.0.

³ <http://bit.ly/2BGWJBU>.

⁴ <http://bit.ly/1a1kSKQ>.

⁵ <http://bit.ly/2Nb9CJS>.

The screenshot shows the Twitter Application Management interface. At the top, there's a header with the Twitter logo and 'Application Management'. Below that, the application name 'MTSW3E' is displayed. There are tabs for 'Details', 'Settings', 'Keys and Access Tokens', and 'Permissions'. A 'Test OAuth' button is visible. The 'Settings' tab is active, showing 'Application Settings'. A note states: 'Keep the "Consumer Secret" a secret. This key should never be human-readable in your application.' The settings include: 'Consumer Key (API Key)', 'Consumer Secret (API Secret)', 'Access Level' (Read and write (modify app permissions)), 'Owner' (MikhailKlassen), and 'Owner ID'. Below this is the 'Application Actions' section with buttons for 'Regenerate Consumer Key and Secret' and 'Change App Permissions'. The 'Your Access Token' section follows, with a note: 'This access token can be used to make API requests on your own account's behalf. Do not share your access token secret with anyone.' It includes fields for 'Access Token', 'Access Token Secret', 'Access Level' (Read and write), 'Owner' (MikhailKlassen), and 'Owner ID'. All secret fields are masked with dots.

Рис. 1.2. Создание нового приложения для Twitter с целью получить все необходимое для авторизации через OAuth и доступа к API на странице по адресу <https://dev.twitter.com/apps>; четыре (затертых) поля OAuth содержат информацию, которую вы будете использовать для обращений к Twitter API

Давайте запустим Jupyter Notebook и приступим к изысканиям. Следуйте за примером 1.1, подставив свои учетные данные в переменные, находящиеся в начале примера, и запустите его, чтобы создать экземпляр Twitter API. В своей работе код использует учетные данные OAuth, чтобы создать объект `auth`, осуществляющий авторизацию через OAuth, и затем передает его классу `Twitter`, выполняющему запросы к Twitter API.

Пример 1.1. Авторизация приложения для доступа к учетной записи в Twitter

```
import twitter

# Посетите http://dev.twitter.com/apps/new, чтобы создать приложение
# и получить учетные данные для этих переменных.
# Эти данные нужно подставить на место пустых строк.
# Дополнительную информацию о реализации OAuth в Twitter вы найдете по адресу:
# https://developer.twitter.com/en/docs/basics/authentication/overview/oauth

CONSUMER_KEY = ''
CONSUMER_SECRET = ''
OAUTH_TOKEN = ''
OAUTH_TOKEN_SECRET = ''

auth = twitter.oauth.OAuth(OAUTH_TOKEN, OAUTH_TOKEN_SECRET,
                           CONSUMER_KEY, CONSUMER_SECRET)

twitter_api = twitter.Twitter(auth=auth)

# Вывод twitter_api ничего не покажет, кроме того,
# что теперь переменная определена

print(twitter_api)
```

Этот пример должен вывести представление объекта `twitter_api`, ясно указывающее, что он был создан, например:

```
<twitter.api.Twitter object at 0x39d9b50>
```

Это показывает, что авторизация через OAuth успешно выполнена и было получено разрешение на обращение к Twitter API.

1.3.3. Исследование актуальных тем

Создав авторизованное подключение к API, можно отправить первый запрос. В примере 1.2 показано, как запросить у Twitter список тем, наиболее актуальных в настоящее время во всем мире, но, если вы горите желанием попробовать некоторые из возможностей, имейте в виду, что API можно легко параметризовать и ограничить темы конкретными регионами. Для ограничения можно использовать идентификатор Yahoo! Geoplanet, получив его от службы Where On Earth (WOE)¹ — программного интерфейса, возвращающего уникальный идентификатор для любого места на Земле (и даже в виртуальном мире). Если

¹ <http://bit.ly/2NHdAJB>.

вы пока не получили такой идентификатор, попробуйте воспользоваться кодом из примера ниже, который получает список актуальных тем как для всего мира, так и только для Соединенных Штатов.

Пример 1.2. Извлечение списка актуальных тем

```
# Всему миру соответствует идентификатор 1 The Yahoo! Where On Earth.  
# См. http://bit.ly/2BGWJBU и  
# http://bit.ly/2MsvwCQ
```

```
WORLD_WOE_ID = 1  
US_WOE_ID = 23424977
```

```
# Обратите внимание на символ подчеркивания перед именем параметра id  
# со строкой, параметризующей запрос. Если передать имя без символа  
# подчеркивания, пакет twitter добавит значение параметра в сам URL,  
# как специальный именованный аргумент.
```

```
world_trends = twitter_api.trends.place(id=WORLD_WOE_ID)  
us_trends = twitter_api.trends.place(id=US_WOE_ID)
```

```
print(world_trends)  
print()  
print(us_trends)
```

Вы должны увидеть неудобочитаемый ответ API — список словарей Python — в отличие от любого сообщения об ошибке. Для примера ниже приводятся усеченные результаты (чуть ниже мы переформатируем ответ, чтобы он был более читаемым):

```
[{'created_at': u'2013-03-27T11:50:40Z', u'trends': [{u'url':  
u'http://twitter.com/search?q=%23MentionSomeoneImportantForYou'...
```

Обратите внимание, что вышеприведенный пример результата содержит URL-адрес тренда в виде поискового запроса, соответствующего хештегу #MentionSomeoneImportantForYou, где %23 — это код символа хештега (#) для использования в URL-адресах. Во всех последующих примерах в этой главе мы будем использовать этот хештег в качестве объединяющей темы. Несмотря на то что пример файла данных с твитами для этого хештега включен в исходный код примеров для книги, вы получите намного больше удовольствия, исследуя тему, которая актуальна в то время, когда вы читаете эту книгу, а не законсервированную нами и уже неактуальную.

Шаблон использования модуля twitter прост и понятен: создайте экземпляр класса Twitter с цепочкой объектов, соответствующей базовому URL-адресу,

а затем вызывайте методы объекта, соответствующие контекстам URL-адресов. Например, `twitter_api.trends.place(_id=WORLD_WOE_ID)` инициирует HTTP GET-запрос `https://api.twitter.com/1.1/trends/place.json?id=1`. Обратите внимание на соответствие URL-адреса и цепочки объектов, созданной с помощью пакета `twitter` для выполнения запроса, и на то, как параметры строки запроса передаются в виде именованных аргументов. Чтобы с помощью пакета `twitter` отправить произвольный запрос в API, обычно достаточно создать запрос таким простым способом, но есть несколько тонких моментов, на которых мы остановимся чуть ниже.

Twitter *ограничивает количество запросов*, которое приложение может послать тому или иному ресурсу API в течение заданного периода времени. Ограничения¹ Twitter подробно описаны в документации (см. рис. 1.3), а кроме того, для удобства каждый отдельный ресурс API определяет свои ограничения. Например, приложение может отправить до 75 запросов на получение списка актуальных тем, подобных тому, что мы только что выполнили, в 15-минутном интервале. За более подробной информацией о том, как действуют ограничения Twitter, обращайтесь к документации². Следуя за примерами в этой главе, вам едва ли удастся превысить ограничение. (В примере 9.17 будут показаны некоторые приемы, оберегающие от превышения ограничений.)



В документации для разработчиков говорится, что результаты запроса к Trends API обновляются не чаще чем раз в пять минут, поэтому бессмысленно прикладывать дополнительные усилия и пытаться запрашивать результаты чаще.

Об этом еще не говорилось явно, но неудобочитаемые результаты, полученные в примере 1.2, выводятся как стандартные структуры данных языка Python. Интерпретатор IPython применит форматированный вывод автоматически, но Jupyter Notebook и стандартный интерпретатор Python не сделают этого за вас. В таких обстоятельствах форматирование можно выполнить с помощью встроенного пакета `json`, как показано в примере 1.3.



JSON³ — это формат обмена данными, с которым вы постоянно будете сталкиваться. Формат JSON предоставляет способ хранения словарей, списков, примитивов, таких как числа и строки, и их комбинаций. Иначе говоря, с помощью JSON можно смоделировать практически любую структуру данных.

¹ <http://bit.ly/2x8c6yq>.

² <http://bit.ly/2Mslpch>.

³ <http://bit.ly/1a1I2IJ>.

Rate Limits: Chart

Rate limits per window

The API rate limits described in this table refer to read-only / GET endpoints. Note that endpoints / resources not listed in the chart default to 15 requests per allotted user. All request windows are 15 minutes in length.

For POST operations, refer to Twitter's Account Limits support page in order to understand the daily limits that apply on a per-user basis.

Endpoint	Resource family	Requests / window (user auth)	Requests / window (app auth)
GET account/verify_credentials	application	75	0
GET application/rate_limit_status	application	180	180
GET favorites/list	favorites	75	75
GET followers/ids	followers	15	15

Рис. 1.3. Ограничения на количество запросов для ресурсов Twitter API указаны в онлайн-документации для каждого вызова API; здесь показана верхняя часть списка вызовов API и соответствующие им ограничения

Пример 1.3. Форматированное отображение JSON-ответов Twitter API

```
import json

print(json.dumps(world_trends, indent=1))
print()
print(json.dumps(us_trends, indent=1))
```

Метод `json.dumps` отформатирует предыдущий ответ Trends API, как показано ниже:

```
[
{
  "created_at": "2013-03-27T11:50:40Z",
  "trends":
  [
    {
      "url": "http://twitter.com/search?q=%23MentionSomeoneImportantForYou",
      "query": "%23MentionSomeoneImportantForYou",
      "name": "#MentionSomeoneImportantForYou",
      "promoted_content": null,
      "events": null
    }
  ]
}
```

```

    },
    ...
  ]
}
]

```

Оба списка актуальных тем легко пробежать взглядом и выявить их общность, но давайте используем для этого множество¹ — структуру данных в языке Python, — которое автоматически решит эту задачу, потому что именно для таких задач были созданы множества. В данном случае под термином *множество* подразумевается математическая структура данных, хранящая неупорядоченную коллекцию уникальных элементов и поддерживающая операции с участием других множеств элементов. Например, операция пересечения двух множеств вернет общие элементы, одновременно присутствующие в обоих множествах; операция объединения вернет все элементы из двух множеств; а операция разности двух множеств действует подобно операции вычитания, когда элементы из одного множества удаляются из другого.

В примере 1.4 показано, как использовать генераторы списков² в Python, чтобы выделить названия актуальных тем из результатов предыдущего запроса, преобразовать списки в множества и найти пересечение множеств для выявления общих элементов. Имейте в виду, что любые два множества тем могут значительно перекрываться друг с другом, в зависимости от того, что на самом деле происходит, когда вы посылаете запрос. Иначе говоря, результаты анализа полностью зависят от запроса и данных, полученных в ответ.



Еще раз напомним, что в приложении В приводится справочник по идиомам языка Python, таким как генераторы списков, знакомство с которыми может вам пригодиться.

Пример 1.4. Вычисление пересечения двух множеств актуальных тем

```

world_trends_set = set([trend['name']
                        for trend in world_trends[0]['trends']])

us_trends_set = set([trend['name']
                     for trend in us_trends[0]['trends']])

common_trends = world_trends_set.intersection(us_trends_set)

print(common_trends)

```

¹ <http://bit.ly/1a1l2Sw>.

² <http://bit.ly/1a1l1hy>.



Обязательно выполните пример 1.4, прежде чем продолжить читать эту главу, чтобы убедиться, что вы в состоянии получать и анализировать данные из Twitter. Можете ли вы объяснить корреляцию между списками актуальных тем в вашей стране и в мире, если она существует?

ТЕОРИЯ МНОЖЕСТВ, ИНТУИТИВНЫЕ ПРЕДСТАВЛЕНИЯ И СЧЕТНАЯ БЕСКОНЕЧНОСТЬ

Операции с множествами могут показаться довольно примитивной формой анализа, тем не менее теория множеств оказала глубокое влияние на математику, образовав основу для многих математических принципов.

Формализацию математического аппарата, положенного в основу теории множеств, обычно приписывают Георгу Кантору (Georg Cantor) и его статье «On a Characteristic Property of All Real Algebraic Numbers»¹ (1874), которая является частью его работы, отвечающей на вопросы, связанные с понятием бесконечности. Для иллюстрации рассмотрим следующий вопрос: имеет ли множество положительных целых чисел большую мощность, чем множество положительных и отрицательных целых чисел?

Хотя интуиция подсказывает, что отрицательных и положительных целых чисел в два раза больше, чем только положительных, в своей работе Кантор показал, что на самом деле мощности этих двух множеств равны! Он показал математически, что оба множества чисел можно отобразить в последовательность, начинающуюся с определенного числа и продолжающуюся в одном направлении до бесконечности, например: $\{1, -1, 2, -2, 3, -3, \dots\}$.

Поскольку числа можно явно перенумеровать и их последовательность не имеет конечной точки, мощности множеств, как говорят, *считно бесконечны*. Иначе говоря, существует определенная последовательность, которой можно было бы следовать детерминистически при наличии достаточного времени для их подсчета.

1.3.4. Поиск твитов

Как оказывается, одним из общих элементов двух множеств актуальных тем является хештег `#MentionSomeoneImportantForYou`, поэтому используем его как основу для поискового запроса, чтобы получить некоторое число твитов для дальнейшего анализа. В примере 1.5 показано, как использовать ресурс `GET search/tweets`² для выполнения конкретного запроса, а также продемонстрировано специальное поле в метаданных результатов поиска, чтобы упростить создание дополнительных запросов для получения большего объема результа-

¹ «Об одном свойстве совокупности всех действительных алгебраических чисел». Русский перевод Ф. А. Медведева в книге: Кантор Георг. Труды по теории множеств. М.: Наука, 1985, с. 18–22; отв. ред. А. Н. Колмогоров, А. П. Юшкевич. — *Примеч. пер.*

² <http://bit.ly/2QtlEf0>.

тов. Описание ресурсов потокового программного интерфейса Twitter Streaming API¹ выходит далеко за рамки этой главы, но он будет представлен в примере 9.9 и с успехом может использоваться во многих ситуациях, когда желательно иметь постоянно обновляемое представление твитов.



Параметры `*args` и `**kwargs` функции, представленной в примере 1.5, — это идиома языка Python, выражающая произвольное число аргументов и именованных аргументов соответственно. Краткий обзор этой идиомы вы найдете в приложении В.

Пример 1.5. Сбор результатов поиска

```
# Присвойте этой переменной актуальную тему или нечто иное.
# Пример запроса ниже был актуальной темой на момент
# написания этих строк и используется на протяжении всей
# этой главы.

q = '#MentionSomeoneImportantForYou'

count = 100

# Импортировать unquote, чтобы предотвратить ошибки кодирования URL в next_results
from urllib.parse import unquote

# См. https://dev.twitter.com/rest/reference/get/search/tweets

search_results = twitter_api.search.tweets(q=q, count=count)

statuses = search_results['statuses']

# Выполнить обход еще пяти пакетов результатов, следующих за текущим
for _ in range(5):
    print('Length of statuses', len(statuses))
    try:
        next_results = search_results['search_metadata']['next_results']
    except KeyError as e: # Если next_results не определена, значит результатов
        # больше нет
        break

    # Создать словарь из next_results, имеющий следующую форму:
    # ?max_id=847960489447628799&q=%23RIPSelena&count=100&include_entities=1
    kwargs = dict([ kv.split('=') for kv in unquote(next_results[1:]).split("&") ])

    search_results = twitter_api.search.tweets(**kwargs)
```

¹ <http://bit.ly/2p7G8hf>.

```
statuses += search_results['statuses']
```

```
# Вывести один из полученных результатов, выбрав один элемент списка...  
print(json.dumps(statuses[0], indent=1))
```



В этом примере мы просто передали хештег программному интерфейсу поиска Search API, но имейте в виду, что он поддерживает ряд мощных операторов¹, позволяющих фильтровать запросы по наличию или отсутствию различных ключевых слов, создателю твита, местоположению, связанному с твитом, и т. д.

По сути, код в примере 1.5 просто повторно посылает запросы к Search API. Одна из проблем, которая может заставить врасплох имеющих опыт работы с другими веб-API (включая версию Twitter API 1), заключается в том, что Search API не поддерживает явно понятия *разбиения на страницы*. Как отмечается в документации с описанием API, это преднамеренное решение и есть несколько веских причин для использования такого *курсорного* решения, в том числе и очень динамичное состояние ресурсов Twitter. Оптимальные методы управления курсором немного отличаются для платформы разработчиков Twitter, потому что Search API предлагает более простой способ навигации по результатам, чем другие ресурсы, например с использованием временных окон.

Результаты поиска содержат специальный узел `search_metadata`, включающий поле `next_results` в строке запроса, обеспечивающее основу для следующего запроса. Если бы для выполнения HTTP-запросов мы не использовали библиотеку, такую как `twitter`, мы могли бы добавить эту предварительно структурированную строку запроса в URL-адрес Search API и снабдить дополнительными параметрами для обработки OAuth. Но так как мы выполняем HTTP-запросы не напрямую, мы должны преобразовать строку запроса в составляющие ее пары ключ/значение и передать их в виде именованных аргументов.

Выражаясь языком Python, мы должны *распаковать* значения из словаря в именованные аргументы и передать их функции. Другими словами, вызов функции внутри цикла `for` в примере 1.5 в конечном счете превращается в вызов:

```
twitter_api.search.tweets(q='%23MentionSomeoneImportantForYou',  
include_entities=1, max_id=313519052523986943)
```

Хотя в исходном коде он выглядит как `twitter_api.search.tweets(**kwargs)`, где `kwargs` — словарь пар ключ/значение.

¹ <http://bit.ly/2CTxv3O>.



Поле `search_metadata` содержит также значение `refresh_url`, которое можно использовать для периодического обновления коллекции результатов новой информацией, появившейся с момента предыдущего запроса.

В следующем примере твита показаны результаты поиска по запросу `#MentionSomeoneImportantForYou`. Найдите минутку, чтобы внимательно изучить их (все). Как уже упоминалось выше, информации в твите больше, чем кажется на первый взгляд. В частности, твит, что приводится ниже, является довольно типичным и в несжатом состоянии содержит более 5 Кбайт кода JSON. Это в 40 раз с лишним больше 140-символьного объема текста, который обычно считается твитом!

```
[
  {
    "contributors": null,
    "truncated": false,
    "text": "RT @hassanmusician: #MentionSomeoneImportantForYou God.",
    "in_reply_to_status_id": null,
    "id": 316948241264549888,
    "favorite_count": 0,
    "source": "Twitter for Android",
    "retweeted": false,
    "coordinates": null,
    "entities": {
      "user_mentions": [
        {
          "id": 56259379,
          "indices": [
            3,
            18
          ],
          "id_str": "56259379",
          "screen_name": "hassanmusician",
          "name": "Download the NEW LP!"
        }
      ],
      "hashtags": [
        {
          "indices": [
            20,
            50
          ],
          "text": "MentionSomeoneImportantForYou"
        }
      ]
    }
  }
],
```

```

"urls": []
},
"in_reply_to_screen_name": null,
"in_reply_to_user_id": null,
"retweet_count": 23,
"id_str": "316948241264549888",
"favorited": false,
"retweeted_status": {
  "contributors": null,
  "truncated": false,
  "text": "#MentionSomeoneImportantForYou God.",
  "in_reply_to_status_id": null,
  "id": 316944833233186816,
  "favorite_count": 0,
  "source": "web",
  "retweeted": false,
  "coordinates": null,
  "entities": {
    "user_mentions": [],
    "hashtags": [
      {
        "indices": [
          0,
          30
        ],
        "text": "MentionSomeoneImportantForYou"
      }
    ],
    "urls": []
  },
  "in_reply_to_screen_name": null,
  "in_reply_to_user_id": null,
  "retweet_count": 23,
  "id_str": "316944833233186816",
  "favorited": false,
  "user": {
    "follow_request_sent": null,
    "profile_use_background_image": true,
    "default_profile_image": false,
    "id": 56259379,
    "verified": false,
    "profile_text_color": "3C3940",
    "profile_image_url_https": "https://si0.twimg.com/profile_images/...",
    "profile_sidebar_fill_color": "95E8EC",
    "entities": {
      "url": {
        "urls": [
          {
            "url": "http://t.co/yRX89YM4J0",

```

```

    "indices": [
      0,
      22
    ],
    "expanded_url": "http://www.datpiff.com/mixtapes-detail.php?id=470069",
    "display_url": "datpiff.com/mixtapes-detail\u2026"
  }
],
},
"description": {
  "urls": []
}
},
"followers_count": 105041,
"profile_sidebar_border_color": "000000",
"id_str": "56259379",
"profile_background_color": "000000",
"listed_count": 64,
"profile_background_image_url_https": "https://si0.twimg.com/profile...",
"utc_offset": -18000,
"statuses_count": 16691,
"description": "#TheseAreTheWordsISaid LP",
"friends_count": 59615,
"location": "",
"profile_link_color": "91785A",
"profile_image_url": "http://a0.twimg.com/profile_images/...",
"following": null,
"geo_enabled": true,
"profile_banner_url": "https://si0.twimg.com/profile_banners/...",
"profile_background_image_url": "http://a0.twimg.com/profile...",
"screen_name": "hassanmusician",
"lang": "en",
"profile_background_tile": false,
"favourites_count": 6142,
"name": "Download the NEW LP!",
"notifications": null,
"url": "http://t.co/yRX89YM4J0",
"created_at": "Mon Jul 13 02:18:25 +0000 2009",
"contributors_enabled": false,
"time_zone": "Eastern Time (US & Canada)",
"protected": false,
"default_profile": false,
"is_translator": false
},
"geo": null,
"in_reply_to_user_id_str": null,
"lang": "en",
"created_at": "Wed Mar 27 16:08:31 +0000 2013",
"in_reply_to_status_id_str": null,

```

```
"place": null,
"metadata": {
  "iso_language_code": "en",
  "result_type": "recent"
},
},
"user": {
  "follow_request_sent": null,
  "profile_use_background_image": true,
  "default_profile_image": false,
  "id": 549413966,
  "verified": false,
  "profile_text_color": "3D1957",
  "profile_image_url_https": "https://si0.twimg.com/profile_images/...",
  "profile_sidebar_fill_color": "7AC3EE",
  "entities": {
    "description": {
      "urls": []
    }
  },
  "followers_count": 110,
  "profile_sidebar_border_color": "FFFFFF",
  "id_str": "549413966",
  "profile_background_color": "642D8B",
  "listed_count": 1,
  "profile_background_image_url_https": "https://si0.twimg.com/profile_...",
  "utc_offset": 0,
  "statuses_count": 1294,
  "description": "i BELIEVE do you? I admire n adore @justinbieber ",
  "friends_count": 346,
  "location": "All Around The World ",
  "profile_link_color": "FF0000",
  "profile_image_url": "http://a0.twimg.com/profile_images/3434...",
  "following": null,
  "geo_enabled": true,
  "profile_banner_url": "https://si0.twimg.com/profile_banners/...",
  "profile_background_image_url": "http://a0.twimg.com/profile_...",
  "screen_name": "LilSalima",
  "lang": "en",
  "profile_background_tile": true,
  "favourites_count": 229,
  "name": "KoKo :D",
  "notifications": null,
  "url": null,
  "created_at": "Mon Apr 09 17:51:36 +0000 2012",
  "contributors_enabled": false,
  "time_zone": "London",
  "protected": false,
  "default_profile": false,
```

```
"is_translator": false
},
"geo": null,
"in_reply_to_user_id_str": null,
"lang": "en",
"created_at": "Wed Mar 27 16:22:03 +0000 2013",
"in_reply_to_status_id_str": null,
"place": null,
"metadata": {
  "iso_language_code": "en",
  "result_type": "recent"
}
},
...
]
```

Твиты обладают самым богатыми метаданными, которые только можно встретить в социальных сетях, и в главе 9 мы рассмотрим некоторые из возможностей, которые они открывают.

1.4. Анализ 140 (или более) символов

Самым исчерпывающим источником информации об объектах платформы Twitter, конечно же, является онлайн-документация, поэтому определенно стоит добавить в закладки страницу «Tweet objects»¹, потому что вам часто придется обращаться к ней в процессе знакомства с устройством твитов. Ни здесь, ни где-либо еще в книге не предпринимается попыток механически повторить онлайн-документацию, однако стоит сделать несколько примечаний, учитывая, что 5 Кбайт информации, содержащейся в твите, для кого-то может оказаться слишком много. Для простоты предположим, что мы извлекли один твит из результатов поиска и сохранили его в переменной `t`. После этого можно вызвать, например, `t.keys()`, чтобы получить список полей верхнего уровня, а инструкция `t['id']` даст доступ к идентификатору твита.



Если вы следуете описанию, выполняя примеры для этой главы в Jupyter Notebook, то рассматриваемый нами твит вы найдете как раз в переменной `t` и сможете в интерактивном режиме исследовать его поля. Текущее рассмотрение предполагает одинаковую организацию, поэтому значения должны соответствовать один к одному.

¹ <http://bit.ly/2OhPimp>.

Вот несколько интересных примечаний.

- Текст твита доступен как `t['text']`:

```
RT @hassanmusician: #MentionSomeoneImportantForYou God.
```

- Сущности, присутствующие в тексте твита, для удобства выделены в словарь `t['entities']`:

```
{
  "user_mentions": [
    {
      "indices": [
        3,
        18
      ],
      "screen_name": "hassanmusician",
      "id": 56259379,
      "name": "Download the NEW LP!",
      "id_str": "56259379"
    }
  ],
  "hashtags": [
    {
      "indices": [
        20,
        50
      ],
      "text": "MentionSomeoneImportantForYou"
    }
  ],
  "urls": []
}
```

- Информация о величине «интереса», который вызвал твит, доступна в полях `t['favorite_count']` и `t['retweet_count']`, хранящих число раз, сколько твит был помещен в закладки или процитирован (ретвитнут) соответственно.
- Если твит является цитатой (ретвитом) другого твита, поле `t['retweeted_status']` представит информацию об исходном твите и его авторе. Имейте в виду, что иногда при цитировании текст твита может изменяться — пользователи могут добавлять свои суждения или иначе манипулировать контентом.
- Поле `t['retweeted']` указывает, был ли данный твит процитирован аутентифицированным пользователем (с использованием авторизованного при-

ложения). Поля, значения которых зависят от конкретного пользователя, в документации разработчиков Twitter называются *зависящими от точки зрения* (perspectival).

- Кроме того, обратите внимание, что с точки зрения API и управления информацией процитированными могут считаться только исходные твиты. То есть поле `retweet_count` сообщает, сколько раз процитирован исходный твит, и должно содержать одно и то же число как в исходном твите, так и во всех последующих его цитатах (ретвитах). Другими словами, цитирование цитаты (ретвит ретвита) считается цитированием исходного твита (ретвитом твита). Первое время это может казаться немного нелогичным, но если вдуматься, цитируя цитату, вы на самом деле цитируете оригинальный твит, полученный через промежуточное звено. Подробный обзор отличий между ретвитами и цитатами твитов вы найдете в разделе «Исследование шаблонов в ретвитах» далее в этой главе.



Распространенной ошибкой является проверка значения поля `retweeted` для определения факта цитирования твита кем-либо. На самом деле для этого нужно проверить наличие в твите узла-обертки `retweeted_status`.

Если у вас возникли какие-либо вопросы, повозитесь с образцом твита и загляните в документацию, прежде чем двигаться дальше. Хорошее знание анатомии твита имеет решающее значение для эффективного анализа данных из Twitter.

1.4.1. Извлечение сущностей из твита

Теперь попробуем выделить сущности и текст из некоторых твитов в структуру данных, удобную для дальнейшего изучения. Код в примере 1.6 извлекает из собранных твитов текст, отображаемые имена и хештеги и использует идиому Python, называемую *двойным* (или *вложенным*) *генератором списков*. Если вы понимаете, как действует простой (одиночный) генератор списка, форматирование кода поможет вам понять, что двойной генератор списков просто возвращает коллекцию значений, порождаемых вложенным циклом. Генераторы списков — очень мощный инструмент, потому что обычно обеспечивают более высокую производительность в сравнении с вложенными списками и имеют понятный (как только вы освоите их) и лаконичный синтаксис.



Генераторы списков широко используются в этой книге, поэтому при необходимости вы можете обратиться к приложению В или официальному руководству по программированию на языке Python¹, чтобы познакомиться с ними поближе.

Пример 1.6. Извлечение из твитов текста, отображаемых имен и хештегов

```
status_texts = [ status['text']
                  for status in statuses ]

screen_names = [ user_mention['screen_name']
                  for status in statuses
                  for user_mention in status['entities']['user_mentions' ]

hashtags = [ hashtag['text']
             for status in statuses
             for hashtag in status['entities']['hashtags' ]

# Получить коллекцию всех слов из всех твитов
words = [ w
          for t in status_texts
          for w in t.split() ]

# Извлечь первые 5 элементов из каждой коллекции...

print(json.dumps(status_texts[0:5], indent=1))
print(json.dumps(screen_names[0:5], indent=1))
print(json.dumps(hashtags[0:5], indent=1))
print(json.dumps(words[0:5], indent=1))
```



Квадратные скобки, следующие за списком или строковым значением, например `status_texts[0:5]`, в языке Python обозначают операцию извлечения среза. Таким способом легко можно извлекать элементы из списков или подстроки из строк. В данном конкретном случае `[0:5]` означает, что из списка `status_texts` извлекаются первые пять элементов (элементы с индексами от 0 до 4). Более подробно об извлечении срезов в языке Python рассказывается в приложении В.

Чтобы было понятнее, какие данные доступны, ниже показаны результаты, полученные этим кодом, — здесь можно видеть текстовое содержимое пяти твитов, отображаемые имена и хештеги:

¹ <http://bit.ly/2otMTZc>.


```
[
  "\u201cKathleenMariee_: #MentionSomeoneImportantForYou @AhhlicksCruise...,
  "#MentionSomeoneImportantForYou My bf @Linkin_Sunrise.",
  "RT @hassanmusician: #MentionSomeoneImportantForYou God.",
  "#MentionSomeoneImportantForYou @Louis_Tomlinson",
  "#MentionSomeoneImportantForYou @Delta_Universe"
]
[
  "KathleenMariee_",
  "AhhlicksCruise",
  "itsravnenn_cx",
  "kandykisses_13",
  "BMOLOGY"
]
[
  "MentionSomeoneImportantForYou",
  "MentionSomeoneImportantForYou",
  "MentionSomeoneImportantForYou",
  "MentionSomeoneImportantForYou",
  "MentionSomeoneImportantForYou"
]
[
  "\u201cKathleenMariee_:",
  "#MentionSomeoneImportantForYou",
  "@AhhlicksCruise",
  ",",
  "@itsravnenn_cx"
]
```

Как и ожидалось, #MentionSomeoneImportantForYou доминирует среди хештегов. В выводе также присутствует несколько часто встречающихся отображаемых имен, которые стоит исследовать.

1.4.2. Исследование твитов и сущностей в них с применением частотного анализа

Практически весь анализ сводится к простому подсчету элементов на некотором уровне, и в этой книге мы в основном будем заниматься подготовкой данных для подсчета и дальнейшим анализом результатов разными способами.

С практической точки зрения подсчет наблюдаемых элементов является отправной точкой почти для всех видов анализа и, следовательно, для статистической фильтрации или манипуляций любого вида с целью выделить слабые

сигналы в искаженных данных. Выше мы просто извлекли первые 5 элементов из каждого неранжированного списка, чтобы получить представление об имеющихся данных, теперь более подробно исследуем эти данные, вычислив частотное распределение и выбрав первые 10 элементов из каждого списка.

Начиная с версии 2.4, в Python появился модуль `collections`¹ с классом `Counter`, который упрощает вычисление частот элементов в коллекциях. Пример 1.7 демонстрирует, как использовать этот класс для вычисления частот и представления результатов в виде ранжированных списков элементов. Одной из веских причин для анализа данных из Twitter является желание узнать, о чем люди говорят *прямо сейчас*. Самый простой способ ответить на этот вопрос — провести обычный частотный анализ, как показано ниже.

Пример 1.7. Частотный анализ слов в твитах

```
from collections import Counter

for item in [words, screen_names, hashtags]:
    c = Counter(item)
    print(c.most_common()[:10]) # Top-10
    print()
```

Вот некоторые результаты, полученные в ходе частотного анализа твитов:

```
[(u'#MentionSomeoneImportantForYou', 92), (u'RT', 34), (u'my', 10),
 (u',', 6), (u'@justinbieber', 6), (u'<3', 6), (u'My', 5), (u'and', 4),
 (u'I', 4), (u'te', 3)]

[(u'justinbieber', 6), (u'Kid_Charliej', 2), (u'Cavillafuerte', 2),
 (u'touchmestyles_', 1), (u'aliceorr96', 1), (u'gymleeam', 1), (u'fienas', 1),
 (u'nayely_1D', 1), (u'angelchute', 1)]

[(u'MentionSomeoneImportantForYou', 94), (u'mentionsomeoneimportantforyou', 3),
 (u'Love', 1), (u'MentionSomeOneImportantForYou', 1),
 (u'MyHeart', 1), (u'bebesito', 1)]
```

В результате частотного анализа мы получили словарь с парами ключ/значение, где ключи соответствуют словам, а значения — их частотам. Теперь отформатируем результаты в таблицу, чтобы упростить их восприятие. Для этого можно установить пакет `prettytable`, выполнив в терминале команду `pip install prettytable`. Этот пакет предлагает удобный способ представле-

¹ <http://bit.ly/2nIrA6n>.

ния данных в виде таблицы фиксированной ширины, которую легко выделить и скопировать в буфер обмена.

Пример 1.8 демонстрирует, как использовать этот пакет для вывода тех же результатов.

Пример 1.8. Использование prettytable для вывода кортежей в форме таблицы

```
from prettytable import PrettyTable

for label, data in (('Word', words),
                   ('Screen Name', screen_names),
                   ('Hashtag', hashtags)):
    pt = PrettyTable(field_names=[label, 'Count'])
    c = Counter(data)
    [ pt.add_row(kv) for kv in c.most_common()[:10] ]
    pt.align[label], pt.align['Count'] = 'l', 'r' # Выравнивание в столбцах
    print(pt)
```

Код в примере 1.8 выводит результаты в виде таблиц, удобных для просмотра, как показано ниже:

Word	Count
#MentionSomeoneImportantForYou	92
RT	34
my	10
,	6
@justinbieber	6
<3	6
My	5
and	4
I	4
te	3

Screen Name	Count
justinbieber	6
Kid_Charliej	2
Cavillafuerte	2
touchmestyles_	1
aliceorr96	1
gymleeam	1
fienas	1
nayely_1D	1
angelchute	1

Hashtag	Count
MentionSomeoneImportantForYou	94
mentionsomeoneimportantforyou	3
NoHomo	1
Love	1
MentionSomeOneImportantForYou	1
MyHeart	1
bebesito	1

При кратком обзоре результатов выясняется одно интересное обстоятельство: пользователь Джастин Бибер (Justin Bieber) занимает высокое место в списке, извлеченном из этого небольшого набора данных, и, учитывая его популярность среди подростков в Twitter, он вполне мог бы быть «наиболее важной персоной» для этой актуальной темы. Впрочем, полученные нами результаты малоубедительны. Также интересно наблюдать высокую частоту последовательности символов <3. Это не что иное, как экранированная форма <3 — стилизованного изображения сердца (повернутого на 90 градусов, подобно другим смайликам), которое служит сокращенной формой слова «люблю». Учитывая характер запроса, появление значения <3 не вызывает удивления, хотя на первый взгляд оно может показаться «шумом».

Конечно, сущности, встречающиеся два и более раз, представляют определенный интерес, но в результатах можно заметить еще кое-что интересное. Например, часто встречается комбинация букв «RT», обозначающая значительное количество ретвитов (подробнее об этом вы узнаете в разделе «Исследование шаблонов в ретвитах» ниже). Наконец, как и следовало ожидать, хештег #MentionSomeoneImportantForYou и пара его вариаций с разным регистром символов занимают лидирующее положение среди хештегов; отсюда можно сделать вывод, что хорошо бы реализовать нормализацию слов, отображаемых имен и хештегов приведением символов к нижнему регистру перед определением частот, потому что такие тонкие отличия неизбежно будут присутствовать в твитах.

1.4.3. Определение лексического разнообразия твитов

Немного более продвинутым анализом, который включает вычисление частот и может применяться к неструктурированному тексту, является оценка *лексического разнообразия*. Математически эта оценка выражается отношением числа

уникальных слов к *общему* числу слов в тексте, которые сами по себе являются элементарными, но важными метриками. Лексическое разнообразие — интересное понятие, родившееся в области межличностных коммуникаций. Оно представляет количественную оценку разнообразия словарного запаса человека или группы людей. Например, представьте, что ваш собеседник постоянно употребляет слова «и прочее» для обобщения информации вместо подкрепления своих слов конкретными примерами или пояснениями. Теперь сравните этого собеседника с тем, кто редко использует слова «и прочее» для обобщения и чаще стремится подкрепить аргументы конкретными примерами. Речь употребляющего слова «и прочее» будет иметь более низкое лексическое разнообразие, чем речь человека, использующего более разнообразный словарный запас, и высока вероятность, что беседа с человеком, который пользуется более богатым словарем, оставит у вас чувство, что он лучше разбирается в обсуждаемом предмете.

Применительно к твитам и подобным онлайн-коммуникациям лексическое разнообразие можно рассматривать как простейшую статистику, помогающую ответить на ряд вопросов, например, насколько широкую или узкую тему обсуждают отдельное лицо или группа. Общая оценка, конечно, представляет определенный интерес, но разбивка твитов на конкретные периоды времени могла бы дать дополнительную информацию, равно как и сравнение различных групп или отдельных лиц. Например, было бы интересно оценить разницу между лексическим разнообразием двух компаний — производителей безалкогольных напитков, таких как Coca-Cola¹ и Pepsi², в качестве исходной точки в исследованиях эффективности их маркетинговых кампаний в социальной сети Twitter.

А теперь, получив представление об использовании такой статистики, как лексическое разнообразие, для анализа текстового контента, такого как твиты, вычислим лексическое разнообразие текстов, отображаемых имен и хештегов в нашем рабочем наборе данных, как показано в примере 1.9.

Пример 1.9. Оценка лексического разнообразия твитов

```
# Функция для вычисления лексического разнообразия
def lexical_diversity(tokens):
    return len(set(tokens))/len(tokens)

# Функция для вычисления среднего числа слов в твите
def average_words(statuses):
```

¹ <http://bit.ly/1a1l5xR>.

² <http://bit.ly/1a1l7pt>.

```
total_words = sum([ len(s.split()) for s in statuses ])
return total_words/len(statuses)

print(lexical_diversity(words))
print(lexical_diversity(screen_names))
print(lexical_diversity(hashtags))
print(average_words(status_texts))
```



До версии Python 3.0 оператор деления (/) применял функцию floor и возвращал целочисленное значение (если только хотя бы один из операндов не являлся значением с плавающей точкой). Если бы мы использовали Python 2.x, нам пришлось бы умножить числитель или знаменатель на 1.0, чтобы избежать ошибок округления.

Вот какие результаты вернул пример 1.9:

```
0.67610619469
0.955414012739
0.0686274509804
5.76530612245
```

Вот несколько наблюдений, сделанных из этих результатов:

- Лексическое разнообразие твитов оценивается величиной около 0,67. Интерпретировать это число можно так: примерно две трети слов уникальны. Также можно сказать, что текст каждого твита содержит около 67% уникальной информации. Учитывая, что среднее количество слов в каждом твите примерно равно шести, получается, что каждый твит содержит четыре уникальных слова. Это вполне согласуется с интуицией в том смысле, что характер хештега #MentionSomeoneImportantForYou предполагает получение в ответ небольшого числа слов. Конечно, значение 0,67 — довольно высокая оценка лексического разнообразия для обычного человеческого общения, но, учитывая характер данных, оно представляется достоверным.
- Однако лексическое разнообразие отображаемых имен еще выше и оценивается значением 0,95. То есть примерно 19 из 20 отображаемых имен являются уникальными. Это наблюдение тоже не лишено смысла, учитывая, что многие ответы на вопрос будут отображаемыми именами и большинство людей будут давать разные ответы для этого хештега.
- Лексическое разнообразие хештегов получилось очень низким и оценивается значением 0,068. Это означает, что в наших результатах очень мало хештегов, отличных от #MentionSomeoneImportantForYou, которые повторяются несколько раз. И снова в этом есть определенный смысл, учитывая, что

большинство ответов короткие и почти никто не вводит хештеги в ответ на приглашение упомянуть кого-то важного для них.

- В среднем твиты содержат очень мало слов, что-то около шести, что имеет смысл, учитывая характер хештега, который предназначен для получения коротких ответов, состоящих всего из нескольких слов.

На данном этапе было бы интересно детальнее рассмотреть некоторые данные и выяснить, можно ли еще что-то получить из них путем проведения более качественного анализа. Учитывая, что в среднем твиты содержат всего шесть слов, маловероятно, что при их составлении пользователи применяли какие-либо сокращения, стараясь остаться в пределах ограничения на число символов, поэтому количество шума в данных должно быть очень низким, и дополнительный частотный анализ мог выявить кое-что интересное.

1.4.4. Исследование шаблонов в ретвитах

Даже притом, что пользовательский интерфейс и многочисленные клиенты Twitter давно используют оригинальный программный интерфейс Retweet API для заполнения таких значений, как `retweet_count` и `retweeted_status`, некоторые пользователи предпочитают делать ретвиты с комментариями¹, что влечет за собой копирование и вставку текста, а также добавление «RT @username» в начало и ссылки на источник «/via @username» в конец.



Анализируя данные из Twitter, вы наверняка пожелаете учесть также информацию в метаданных и использовать эвристики для анализа таких общепринятых строк символов, как «RT @username» или «/via @username» при изучении ретвитов, чтобы обеспечить максимальную эффективность анализа. Более подробное обсуждение отличий использования оригинального Retweet API от «цитирования» твитов с применением соглашений для ссылки на источник вы найдете в разделе «Поиск пользователей, выполнивших ретвит».

На данном этапе хорошим упражнением было бы продолжить анализ данных и определить, был ли конкретный твит многократно ретвитнут или просто было сделано много «разовых» ретвитов. Чтобы найти наиболее популярные ретвиты, просто выполним итерации по обновлениям статусов и сохраним количество ретвитов, инициатора ретвита и текст ретвита, если обновление статуса является ретвитом. В примере 1.10 показано, как извлечь эти значения с использованием генератора списка и сортировкой по количеству ретвитов.

¹ <http://bit.ly/1a1l7FZ>.

Пример 1.10. Поиск наиболее популярных ретвитов

```
retweets = [
    # Сохранить кортеж с тремя значениями...
    (status['retweet_count'],
     status['retweeted_status']['user']['screen_name'],
     status['text'])

    # ...для каждого статуса...
    for status in statuses

    # ...соответствующего этому условию
    if 'retweeted_status' in status.keys()
]

# Выбрать первые 5 результатов из сортированного списка
# и вывести каждый элемент, имеющийся в кортеже

pt = PrettyTable(field_names=['Count', 'Screen Name', 'Text'])
[ pt.add_row(row) for row in sorted(retweets, reverse=True)[:5] ]
pt.max_width['Text'] = 50
pt.align = 'l'
print(pt)
```

Результаты примера 1.10 выглядят довольно интересными:

Count	Screen Name	Text
23	hassanmusician	RT @hassanmusician: #MentionSomeoneImportantForYou God.
21	HSweethearts	RT @HSweethearts: #MentionSomeoneImportantForYou my high school sweetheart ❤️
15	LosAlejandro_	RT @LosAlejandro_: ¿Nadie te menciona en "#MentionSomeoneImportantForYou"? JAJAJAJAJAJAJA JAJAJAJAJAJAJAJAJAJAJAJAJAJAJAJAJAJA Ven, ...
9	SCOTTSUMME	RT @SCOTTSUMME: #MentionSomeoneImportantForYou My Mum. Shes loving, caring, strong, all in one. I love her so much ❤️❤️❤️
7	degrassihaaha	RT @degrassihaaha: #MentionSomeoneImportantForYou I can't put every Degrassi cast member, crew member, and writer in just one tweet....

Список возглавляет «God» (Бог), за ним следует «my high school sweetheart» (моя школьная любовь), а на четвертом месте стоит «My Mum» (моя мама). Ни один из пяти первых пунктов списка не соответствует аккаунтам пользователей Twitter, хотя по результатам предыдущего анализа мы могли бы предположить это (за исключением @justinbieber). Продолжив исследование результатов, ниже

в списке можно обнаружить упоминания конкретных пользователей, но выборка, которую мы взяли для исследования, настолько мала, что никаких тенденций не выявляется. Поиск по более крупной выборке, вероятно, даст несколько упоминаний пользователей с частотой больше одного, что было бы интересно для дальнейшего анализа. Возможности для исследований ничем не ограничены, и я уверен, что вам не терпится попробовать некоторые собственные запросы.



В конце этой главы предлагается несколько упражнений. Не забудьте также заглянуть в главу 9, чтобы почерпнуть новые идеи: она включает более двух десятков готовых рецептов.

Прежде чем продолжить, стоит отметить, что в результатах поиска вполне могут отсутствовать (и почти наверняка отсутствуют, учитывая относительно низкие частоты ретвитов, наблюдаемые в этом разделе) оригинальные твиты, которые были ретвитнуты. Например, самый популярный ретвит в выборке принадлежит пользователю с именем @hassanmusician, он был ретвитнут 23 раза. Однако тщательное исследование показывает, что в результатах поиска присутствует только 1 из 23 ретвитов. Ни оригинального твита, ни любого другого из 22 ретвитов нет в наборе данных. Это не создает каких-то особых проблем, но может возникнуть вопрос: кто эти 22 других ретвитера?

Возможность получить ответ на этот вопрос представляет большую ценность, потому что позволяет взять контент, представляющий некоторую идею, такую как «God» (Бог) в данном случае, и выявить группу пользователей, разделяющих те же чувства или интересы. Как уже упоминалось выше, удобный способ моделирования данных с участием людей и вещей, которые их интересуют, называется *графом интересов*; это основная структура данных, лежащая в основе анализа в главе 8. В отношении этих пользователей можно предположить, что они являются духовными или религиозными людьми, и дальнейший анализ их твитов мог бы подтвердить этот вывод. В примере 1.11 показано, как найти таких с помощью программного интерфейса `GET statuses/retweets/:id1`.

Пример 1.11. Поиск пользователей, ретвитнувших статус

Возьмите идентификатор оригинального твита из узла `retweeted_status` ретвита
и вставьте его на место примерного значения, указанного в тексте книги

```
_retweets = twitter_api.statuses.retweets(id=317127304981667841)
print([r['user']['screen_name'] for r in _retweets])
```

¹ <http://bit.ly/2BHBEaq>.

Дальнейший анализ группы пользователей, ретвитнувших этот конкретный статус, с целью выяснения их религиозной или духовной принадлежности мы оставляем вам в качестве самостоятельного упражнения.

1.4.5. Визуализация частот с помощью гистограмм

Jupyter Notebook обладает замечательной возможностью создания и вставки высококачественных и настраиваемых графиков в ходе интерактивного сеанса работы. В частности, пакет `matplotlib` и другие инструменты для научных вычислений, которые доступны Jupyter Notebook, обладают довольно широкими возможностями и способны генерировать сложные фигуры малыми усилиями, от вас требуется только освоить основные операции.

Для иллюстрации возможностей `matplotlib` построим несколько графиков на основе имеющихся у нас данных. Для начала рассмотрим график, отображающий результаты в переменной `words`, полученные в примере 1.9. С помощью класса `Counter` мы без труда можем сгенерировать отсортированный список кортежей, каждый из которых содержит пару (`word`, `frequency`); по оси X будем откладывать индексы кортежей, а по оси Y — частоты слов в этих кортежах. Было бы непрактично пытаться откладывать сами слова по оси X, хотя именно для этого она и предназначена. На рис. 1.4 показан график изменения частот

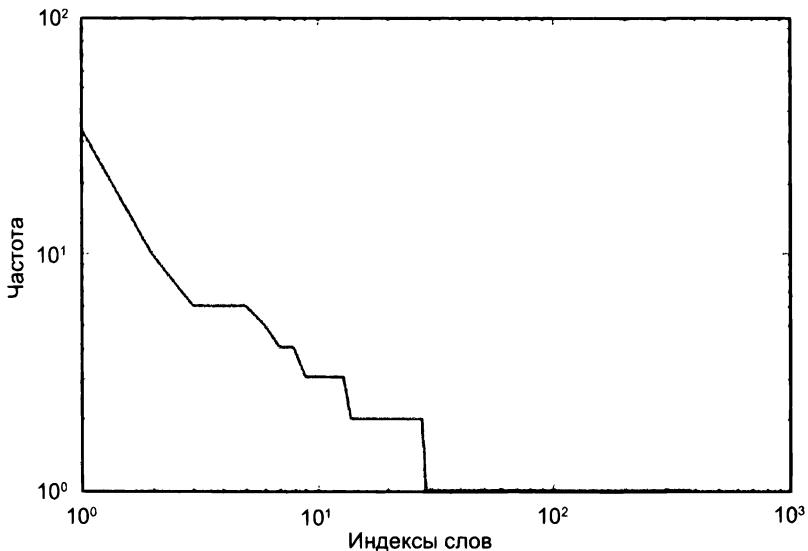


Рис. 1.4. График, отображающий отсортированные частоты слов, вычисленные в примере 1.8

слов, полученный на основе данных, представленных выше в виде таблицы, следующей за примером 1.8. Значения на оси Y соответствуют количеству появлений слова. На оси X отсутствуют метки, представляющие слова, но значения вдоль оси X были отсортированы так, чтобы сделать более очевидной связь между частотами слов. Каждая ось приведена к логарифмическому масштабу для сглаживания кривой. График можно сгенерировать непосредственно в Jupyter Notebook, если ввести код из примера 1.12.



В 2014 году флаг `--pylab`, использовавшийся при запуске IPython/Jupyter Notebook, был объявлен устаревшим. В том же году Фернандо Перес (Fernando Pérez) начал работу над проектом Jupyter. Чтобы включить поддержку встраиваемых графиков в Jupyter Notebook, добавьте в ячейку с кодом команду `%matplotlib`:

```
%matplotlib inline
```

Пример 1.12. Построение графика частот слов

```
import matplotlib.pyplot as plt
%matplotlib inline

word_counts = sorted(Counter(words).values(), reverse=True)

plt.loglog(word_counts)
plt.ylabel("Freq")
plt.xlabel("Word Rank")
```

График частот достаточно прост и понятен, но, возможно, было бы полезно сгруппировать слова по диапазонам частот, чтобы, например, можно было ответить на вопрос, сколько слов имеют частоту от 1 до 5, от 5 до 10, от 10 до 15 и так далее. Именно для таких случаев предназначены *гистограммы*² — они обеспечивают удобный способ визуализации табличных частот в виде ряда смежных столбиков, каждый из которых представляет значения, попадающие в соответствующий диапазон. На рис. 1.5 и 1.6 показаны гистограммы табличных данных, полученных на примерах 1.8 и 1.10 соответственно. Гистограммы не имеют меток вдоль оси X, которые показывали бы, какие слова соответствуют каждой группе, но на самом деле это не является их целью. Гистограмма дает общее представление о распределении частот, где ось X соответствует диапазонам слов, каждое из которых имеет частоту, попадающую в определенный диапазон, и ось Y соответствует количеству всех слов в этом диапазоне.

¹ <http://bit.ly/2nrbbkQ>.

² <http://bit.ly/1a1l6Sk>.

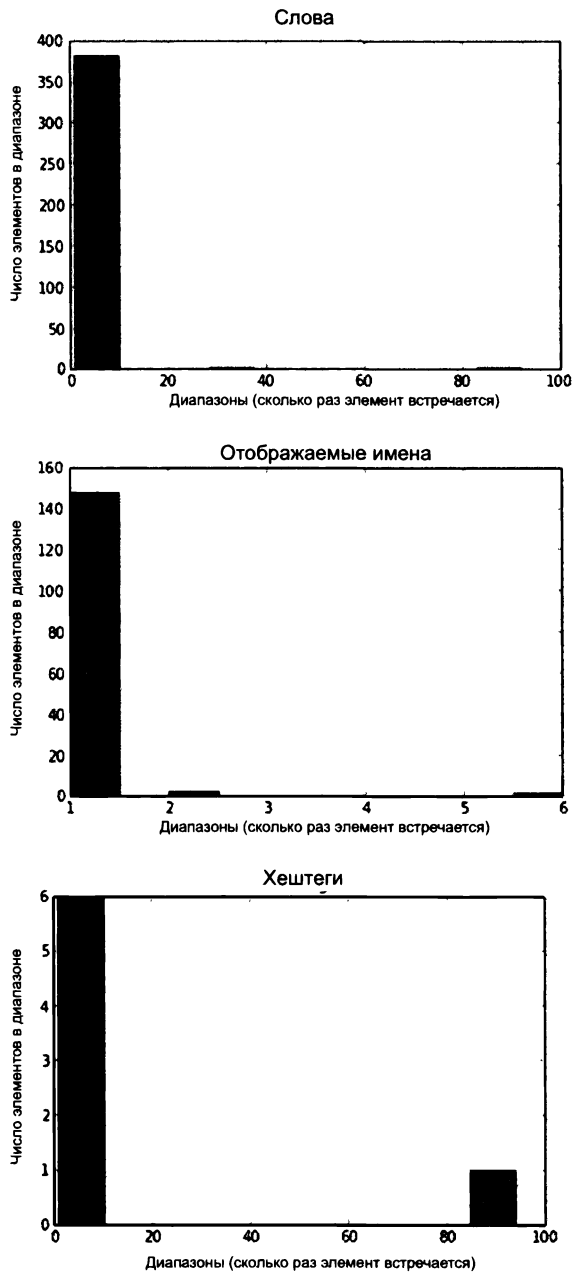


Рис. 1.5. Гистограммы, отображающие результаты частотного анализа слов, отображаемых имен и хештегов, сгруппированных по частоте



Рис. 1.6. Гистограмма частот ретвитов

Интерпретируя рис. 1.5, вернитесь к соответствующим табличным данным и обратите внимание на большое количество слов, отображаемых имен и хештегов, которые имеют низкую частоту и редко появляются в тексте; объединив все эти низкочастотные термины в диапазон «все слова с частотой от 1 до 10», мы увидим, что они составляют большую часть текста. Более конкретно, мы видим, что почти все слова принадлежат диапазону частот от 1 до 10, которому соответствует большой синий прямоугольник, и лишь два слова имеют гораздо более высокие частоты: «#MentionSomeoneImportantForYou» и «RT» с частотами 34 и 92 в наших данных.

Аналогично, интерпретируя рис. 1.6, можно заметить, что часто цитируемых твитов очень немного, тогда как большинство, представленное самым большим синим прямоугольником слева на гистограмме, цитируется только раз.

Код, генерирующий эти гистограммы в Jupyter Notebook, приводится в примерах 1.13 и 1.14. Найдите время, чтобы поближе познакомиться с возможностями `matplotlib` и других инструментов для научных вычислений.



Установка инструментов для научных вычислений, таких как `matplotlib`, может оказаться утомительной задачей из-за некоторых динамически загружаемых библиотек в их цепочках зависимостей, и степень сложности может варьироваться в зависимости от версии и операционной системы. Поэтому мы настоятельно рекомендуем воспользоваться преимуществами виртуальной машины для этой книги, которая описывается в приложении А, если вы ее еще не установили.

Пример 1.13. Создание гистограмм слов, отображаемых имен и хештегов

```
for label, data in (('Words', words),
                   ('Screen Names', screen_names),
                   ('Hashtags', hashtags)):

    # Создать частотный словарь для каждого набора данных
    # и построить гистограммы
    c = Counter(data)
    plt.hist(c.values())

    # Добавить подпись для оси Y...
    plt.title(label)
    plt.ylabel("Number of items in bin")
    plt.xlabel("Bins (number of times an item appeared)")

    # ...и вывести гистограмму как новую фигуру
    plt.figure()
```

Пример 1.14. Создание гистограммы с частотами ретвитов

```
# Использование подчеркиваний при извлечении значений
# из кортежей является идиоматическим способом их игнорирования

counts = [count for count, _, _ in retweets]

plt.hist(counts)
plt.title("Retweets")
plt.xlabel('Bins (number of times retweeted)')
plt.ylabel('Number of tweets in bin')

print(counts)
```

1.5. Заключительные замечания

В этой главе была представлена социальная сеть Twitter — успешная технологическая платформа, быстро приобретающая популярность благодаря ее способности удовлетворять некоторые фундаментальные человеческие желания, связанные с общением, любопытством и самоорганизующимся поведением, возникшим из ее хаотической сетевой динамики. Пример кода в этой главе познакомил вас с программным интерфейсом Twitter API, продемонстрировал, как легко (и интересно) использовать Python для интерактивного изучения и анализа данных Twitter, и предоставил некоторые начальные шаблоны, которые вы можете использовать для анализа твитов. В начале главы мы узнали, как создать аутентифицированное соединение, а затем прошли через серию примеров, иллюстрирующих, как выявить актуальные темы для конкретных стран и регионов, как искать твиты, которые могут быть интересны, и как анализировать

эти твиты с применением некоторых простых, но эффективных методов на основе частотного анализа и простой статистики. Как оказалось, даже произвольно выбранную актуальную тему можно проанализировать множеством интересных способов.



В главе 9 вы найдете ряд рецептов для Twitter, охватывающих широкий спектр тем, от сбора и анализа твитов до эффективного использования пространства для хранения твитов и методов анализа подписчиков, или фолловеров (followers).

Один из основных аналитических выводов этой главы, который вы должны усвоить, заключается в том, что первым шагом к любому значимому количественному анализу обычно является подсчет. Базовый частотный анализ, несмотря на свою простоту, является мощным инструментом, и его нельзя упускать из виду только потому, что он настолько очевиден; кроме того, многие другие виды статистического анализа зависят от его результатов. Более того, именно из-за простоты и очевидности выполнять частотный анализ и вычислять такие метрики, как лексическое разнообразие, следует как можно раньше и чаще. Часто, но не всегда, результаты простейших методов по информативности могут соперничать с результатами более сложных видов анализа. Применительно к данным из вселенной Twitter эти скромные приемы нередко могут дать вам возможность сделать первый шаг на долгом пути к ответу на вопрос: «О чем люди говорят сейчас?» Многие из нас хотели бы знать это, не так ли?



Исходный код примеров для этой и всех других глав доступен на GitHub¹ в удобном формате Jupyter Notebook, который вы можете опробовать, не покидая веб-браузера.

1.6. Упражнения

- Заложите на этой странице закладку и познакомьтесь с содержимым справочника по программному интерфейсу «API reference index»². Особое внимание уделите разделам «REST API»³ и «API objects»⁴.

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² <http://bit.ly/2Nb9CJS>.

³ <http://bit.ly/2nTNdF>.

⁴ <http://bit.ly/2oL2EdC>.

- Если вы еще не сделали этого, познакомьтесь поближе с IPython и Jupyter Notebook — более удобными альтернативами традиционному интерпретатору Python. В вашей карьере в сфере анализа данных в социальных сетях они сэкономят вам немало сил и времени.
- Если у вас есть аккаунт Twitter с достаточно большим количеством твитов, запросите свой архив твитов из настроек¹ и проанализируйте его. В этом архиве вы найдете файлы в формате JSON, организованные по периодам времени. За дополнительными подробностями обращайтесь к файлу `README.txt`, который также будет включен в архив. Попробуйте ответить на вопросы: какие слова чаще всего встречаются в ваших твитах? Кого из пользователей вы ретвитите чаще всего? Сколько ваших твитов было ретвитнуто (и почему, по вашему мнению, были ретвитнуты именно они)?
- Потратьте некоторое время на изучение Twitter REST API с помощью инструмента командной строки Twurl². В этой главе мы использовали пакет `twitter` для Python, однако консоль может быть очень ценным инструментом для изучения API, влияния параметров и многого другого.
- Завершите упражнение по определению духовной или религиозной принадлежности пользователей, которые ретвитнули статус со словом «God» (Бог), как упоминание кого-то важного для них, или выполните процедуру, описанную в этой главе, анализа актуальной темы по вашему выбору. Исследуйте дополнительные возможности поиска³, помогающие уточнить запрос.
- Исследуйте Yahoo! GeoPlanet Where On Earth ID API⁴, чтобы получить возможность сравнивать и сопоставлять темы, актуальные для разных стран и регионов.
- Познакомьтесь поближе с `matplotlib`⁵ и узнайте, как создавать двух- и трехмерные диаграммы⁶ в Jupyter Notebook.
- Исследуйте и попробуйте выполнить несколько упражнений из главы 9.

¹ <http://bit.ly/1a1b8D>.

² <http://bit.ly/2NIQItE>.

³ <http://bit.ly/2xIEHzB>.

⁴ <http://bit.ly/2MswCQ>.

⁵ <http://bit.ly/1a17Wwv>.

⁶ <http://bit.ly/1a1lccP>.

1.7. Онлайн-ресурсы

Ниже приводится список ссылок, упоминавшихся в этой главе, которые могут оказаться полезными для вас:

- Создание двух- и трехмерных диаграмм в Jupyter Notebook¹.
- Служебные команды IPython².
- json.org³.
- Генераторы списков в Python⁴.
- Официальное руководство по языку Python⁵.
- OAuth⁶.
- Документация с описанием Twitter API⁷.
- Пределы в версии Twitter API 1.1⁸.
- Соглашение и политика для разработчиков Twitter⁹.
- Документация с описанием поддержки OAuth в Twitter¹⁰.
- Операторы Twitter Search API¹¹.
- Поточковый программный интерфейс Twitter Streaming API¹².
- Условия обслуживания Twitter¹³.
- Twurl¹⁴.
- Yahoo! GeoPlanet Where On Earth ID¹⁵.

¹ <http://bit.ly/1a1lccP>.

² <http://bit.ly/2nII3ce>.

³ <http://bit.ly/1a1l2IJ>.

⁴ <http://bit.ly/2otMTZc>.

⁵ <http://bit.ly/2oLozBz>.

⁶ <http://bit.ly/1a1kZWN>.

⁷ <http://bit.ly/1a1kSKQ>.

⁸ <http://bit.ly/2MsLpch>.

⁹ <http://bit.ly/2MSrrys>.

¹⁰ <http://bit.ly/2NawA3v>.

¹¹ <http://bit.ly/2xkjW7D>.

¹² <http://bit.ly/2Qzcdvd>.

¹³ <http://bit.ly/1a1kWKb>.

¹⁴ <http://bit.ly/1a1kZq1>.

¹⁵ <http://bit.ly/2NHdAJB>.

2

Facebook: анализ фан-страниц, исследование дружественных связей и многое другое

В этой главе мы подключимся к платформе Facebook через ее (Social) Graph API и рассмотрим некоторые из многочисленных возможностей. Facebook — одна из самых обширных социальных сетей и является чем-то вроде швейцарского армейского ножа. Кроме того, более половины из двух миллиардов пользователей этой сети¹ проявляют активность каждый день, обновляя статусы, размещая фотографии, обмениваясь сообщениями, беседуя в режиме реального времени, играя в игры, совершая покупки и почти все, что только можно представить. С точки зрения анализа социальных сетей данные, которые Facebook хранит о людях, группах и продуктах, представляют очень большой интерес, потому что Facebook API предлагает невероятные возможности для получения информации (самый ценный товар в мире) и сбора ценных идей. С другой стороны, большие возможности накладывают большую ответственность, поэтому в Facebook реализован самый сложный онлайн-контроль конфиденциальности², который когда-либо видел мир, чтобы помочь защитить своих пользователей от посягательств.

Стоит отметить, что, хотя социальная сеть Facebook провозгласила себя социальным графом, она постепенно превращается также в ценный граф интересов, потому что позволяет устанавливать отношения между людьми и предметами,

¹ Статистика использования интернета (<http://bit.ly/1a1ljf8>) показывает, что в 2017 году население земного шара начитывало примерно 7,5 миллиарда человек, а интернетом пользовалось примерно 3,9 миллиарда.

² <http://on.fb.me/1a1llg9>.

которые их интересуют, и выражать реакцию (например, кликнув Like (Нравится)). В связи с этим все чаще можно услышать, что Facebook позиционируется как «социальный граф интересов». В целом можно смело утверждать, что графы интересов неявно присутствуют и доступны для извлечения в большинстве источников социальных данных. Например, в главе 1 было показано, что Twitter действительно является графом интересов благодаря его асимметричной модели «следования» (или, другими словами, «заинтересованности») в отношениях между людьми и другими людьми, местами или вещами. Идея представления Facebook как графа интересов протянется красной нитью через эту главу, а подробнее об извлечении графа интересов из социальных данных мы поговорим в главе 8.

Далее в этой главе предполагается, что у вас есть активный аккаунт в Facebook¹, который необходим для получения доступа к Facebook API. В 2015 году в Facebook API были внесены изменения², ограничившие доступ к данным третьим лицам. Например, вы больше не сможете получить доступ к обновлениям статуса или списку интересов вашего друга через программный интерфейс. Эти изменения обусловлены соображениями конфиденциальности. Из-за них в этой главе больше внимания будет уделено использованию Facebook API для измерения вовлеченности пользователей на публичных страницах³, например, созданных компаниями или знаменитостями. Но для доступа даже к этим данным требуется все больше привилегий, и вам, возможно, придется запросить одобрение в Facebook, чтобы получить доступ к некоторым из имеющихся возможностей.



Загрузите последний исправленный исходный код примеров для этой (и любой другой) главы, доступный по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Воспользуйтесь также преимуществами виртуальной машины, описанными в приложении А, чтобы получить максимальное удовольствие от опробования примеров кода.

2.1. Обзор

Так как это уже вторая глава в книге, здесь мы рассмотрим вещи, более сложные, чем в главе 1, но все еще доступные для широкой аудитории. В этой главе вы познакомитесь с:

¹ <http://on.fb.me/1a1lkcd>.

² <http://tcrn.ch/2zFetfo>.

³ В терминологии Facebook — фан-страницах, или пабликах. — *Примеч. пер.*

- Facebook Graph API и узнаете, как посылать запросы этому программному интерфейсу;
- протоколом Open Graph и узнаете, как он связан с социальным графом в Facebook;
- возможностью программного доступа к публичным страницам, созданным, например, компаниями или знаменитостями;
- приемами извлечения основных социальных показателей, таких как количество лайков, комментариев и ссылок для оценки вовлеченности аудитории;
- управлением данными с использованием pandas DataFrames и последующей визуализацией результатов.

2.2. Facebook Graph API

Платформа Facebook — зрелый, надежный и хорошо документированный интерфейс к наиболее полному как с точки зрения широты, так и глубины и хорошо организованному хранилищу информации. В ширину она охватывает около четверти всего населения мира, а в глубину включает большой объем сведений о любом из ее пользователей. В отличие от Twitter, где используется асимметричная модель дружбы, которая открыта и основана на следовании за другими пользователями без всякого согласия, в Facebook модель дружбы симметрична и требует взаимного согласия пользователей, чтобы они могли видеть взаимодействия и действия друг друга.

Кроме того, в отличие от Twitter, где практически все общение, кроме личных сообщений, происходит посредством публичных статусов, Facebook обеспечивает гораздо более тонкое управление конфиденциальностью, и дружественные отношения могут организовываться и поддерживаться в виде списков с различными уровнями видимости, доступными другу по любому конкретному виду деятельности. Например, вы можете поделиться ссылкой или фотографией только с определенным списком друзей, а не со всей вашей социальной сетью.

Извлекать данные из Facebook для анализа можно, только зарегистрировав приложение и используя его как точку входа в платформу разработчиков Facebook. Кроме того, такому приложению будут доступны только те данные, доступ к которым явно разрешен пользователем. Например, как разработчик приложения для Facebook, вы также будете выступать в роли пользователя, входящего в приложение, и приложение будет иметь доступ только к тем данным, доступ

к которым вы разрешите явно. Как пользователь, вы можете думать о приложении как об одном из ваших друзей в Facebook: вы сможете полностью контролировать доступность тех или иных данных для приложения и закрыть доступ к ним в любой момент. Документ с описанием политики платформы Facebook¹ должен прочитать любой разработчик Facebook, потому что в нем исчерпывающе описываются права и обязанности всех пользователей Facebook, а также дух и буква закона для разработчиков Facebook. Если вы еще не сделали этого, воспользуйтесь моментом, ознакомьтесь с политикой разработчиков Facebook и добавьте закладку на домашнюю страницу разработчиков Facebook² — единую точку доступа к платформе Facebook и ее документации. Кроме того, имейте в виду, что программный интерфейс может меняться. Вследствие неумолимой заботы о безопасности и конфиденциальности ваши привилегии, доступные на время экспериментов с платформой Facebook, будут ограничены. Чтобы получить доступ к некоторым возможностям и конечным точкам API, может потребоваться отправить приложение на проверку и утверждение³. Пока ваше приложение будет соблюдать условия обслуживания⁴, у вас все должно быть в порядке.



Как у разработчика, анализирующего собственный аккаунт, у вас не должно возникнуть проблем с доступом к данным вашего аккаунта. Но будьте осторожны и не стремитесь создать *приложение*, которое пытается получить больше данных, чем необходимо для выполнения его задачи, потому что пользователь может не доверять вашему приложению такой уровень привилегий (и правильно сделает).

Далее в этой главе мы будем обращаться к платформе Facebook программным способом, однако Facebook предоставляет ряд полезных инструментов для разработчиков⁵, включая приложение Graph API Explorer⁶, которое мы используем для начального знакомства с социальным графом. Приложение предлагает простой и понятный способ опроса социального графа, и как только вы усвоите особенности его работы, мы переведем запросы на язык Python для их автоматизации, и дальнейшая обработка будет осуществляться естественным путем. Несмотря на то что в рамках обсуждения мы будем использовать Graph API, вы можете извлечь дополнительную выгоду, прочитав исчерпывающее введение в API Graph⁷.

¹ <http://bit.ly/1a1m3C>.

² <http://bit.ly/1a1m3Q>.

³ <http://bit.ly/2vDb2B1>.

⁴ <http://on.fb.me/1a1IMXM>.

⁵ <http://bit.ly/1a1nVf>.

⁶ <http://bit.ly/2jd5Xdq>.

⁷ <http://bit.ly/1a1lobU>.



Обратите внимание, что Facebook прекратил поддержку языка запросов Facebook Query Language¹ (FQL). Начиная с 8 августа 2016 года запросы FQL больше не обрабатываются. Теперь разработчики должны выполнять вызовы к Graph API. Для тех, кто раньше уже создал приложение, использующее FQL, Facebook выпустил инструмент обновления API Upgrade Tool², который можно использовать для обновления таких приложений.

2.2.1. Знакомство с Graph API

Как нетрудно догадаться, социальный граф Facebook — это обширный граф³ — структура данных, представляющая социальные связи и состоящая из узлов и связывающих их ребер. Graph API предоставляет основные средства взаимодействия с социальным графом, и лучший способ познакомиться с этим программным интерфейсом — потратить несколько минут на эксперименты с приложением Graph API Explorer⁴.

Важно отметить, что Graph API Explorer не отличается ничем особенным. Кроме возможности ввода и проверки токена доступа, это самое обычное приложение для Facebook, использующее тот же программный интерфейс, что и любое другое приложение. С другой стороны, приложение Graph API Explorer удобно использовать, когда у вас есть определенный токен OAuth, связанный с определенным набором разрешений для разрабатываемого приложения, и вы хотите выполнить несколько запросов для проверки своих предположений или для отладки. Мы еще вернемся к этой идее чуть позже, когда займемся организацией программного доступа к Graph API. На рис. 2.1, 2.2 и 2.3 показана последовательность запросов к Graph API, выполняемых в результате щелчка на значке с символом «плюс» (+) и добавления связей и полей. Отметим некоторые элементы на этих рисунках:

Токен доступа

Токен доступа (access token), отображаемый в приложении, — это токен OAuth⁵, предоставляемый зарегистрированному пользователю для удобства; именно этот токен OAuth потребуется приложению для доступа к данным. Мы будем использовать его на протяжении всей этой главы, но вы можете

¹ <http://bit.ly/1a1lmRd>.

² <http://bit.ly/2MGU7Z9>.

³ <http://bit.ly/1a1loIX>.

⁴ <http://bit.ly/2jd5Xdq>.

⁵ <http://bit.ly/1a1kZWN>.

также ознакомиться с приложением Б, где дается краткий обзор OAuth, включая сведения об особенностях реализации OAuth в Facebook и приемах получения токена доступа. Как уже упоминалось в главе 1, если это ваше первое знакомство с OAuth, на данный момент вам достаточно знать, что это протокол открытой авторизации, ставший стандартом для социальных сетей. Проще говоря, OAuth — это средство, с помощью которого пользователи могут разрешать сторонним приложениям получать доступ к данным в их учетных записях без необходимости передавать конфиденциальную информацию, например пароль.

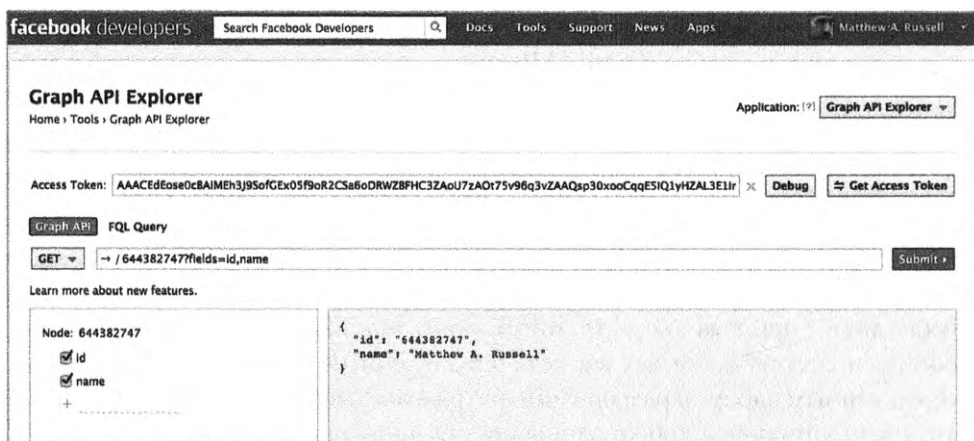


Рис. 2.1. Использование приложения Graph API Explorer для получения узла в социальном графе



Дополнительные сведения об использовании протокола OAuth 2.0, которые понадобятся при создании приложения, требующего авторизации произвольного пользователя для доступа к данным учетной записи, вы найдете в приложении Б.

Идентификаторы узлов

Основой запроса является узел с идентификатором (ID) «644382747», который соответствует человеку с именем «Matthew A. Russell» (Мэтью А. Рассел) и был загружен для текущего пользователя Graph Explorer, выполнившего вход. Значения «id» и «name» называются *полями* узла. Основой запроса мог бы быть любой другой узел, и, как мы скоро увидим, без всякого труда можно выполнять обход графа и запрашивать другие узлы (которые могут представлять людей или неодушевленные существа, такие как книги или телешоу).

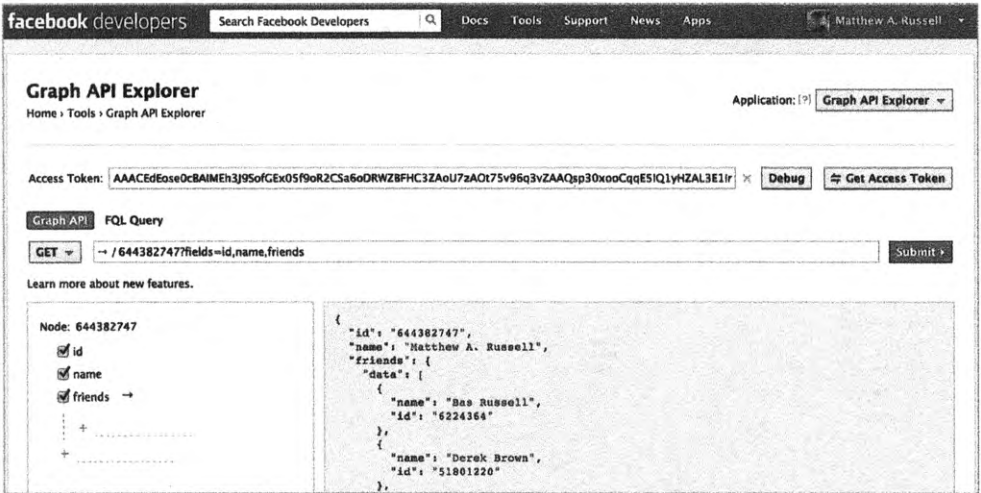


Рис. 2.2. Использование приложения Graph API Explorer для последовательного получения узла и его связей с друзьями. Имейте в виду, что для доступа к некоторым данным могут потребоваться дополнительные привилегии, а изолированным приложениям доступен очень ограниченный набор данных. За годы существования в Facebook внесли много изменений в их политику конфиденциальности

Связи

Запрос можно изменить с помощью связи «friends» (друзья), как показано на рис. 2.2, щелкнув на значке с символом + и выбрав в контекстном меню «connections» (связи) пункт «friends» (друзья). В разделе «friends», отображаемом в консоли, перечислены узлы, связанные с исходным узлом запроса. В данный момент вы можете щелкнуть на значении в поле «id» любого из этих узлов и инициировать запрос с этим конкретным узлом в качестве основы. Выразаясь языком теории графов, вы получили *эго-граф* (ego graph), с субъектом (*эго*) в его логическом центре, который связан с окружающими его узлами. Если нарисовать эго-граф, он будет напоминать колесо со спицами.

«Лайки»

Еще больше изменить запрос можно, добавив связи с «лайками» каждого из ваших друзей, как показано на рис. 2.3. Однако теперь эти данные считаются конфиденциальными и приложения больше не имеют доступа к «лайкам» ваших друзей. В Facebook были внесены изменения в API, ограничивающие круг информации, к которой приложения могут получить доступ.

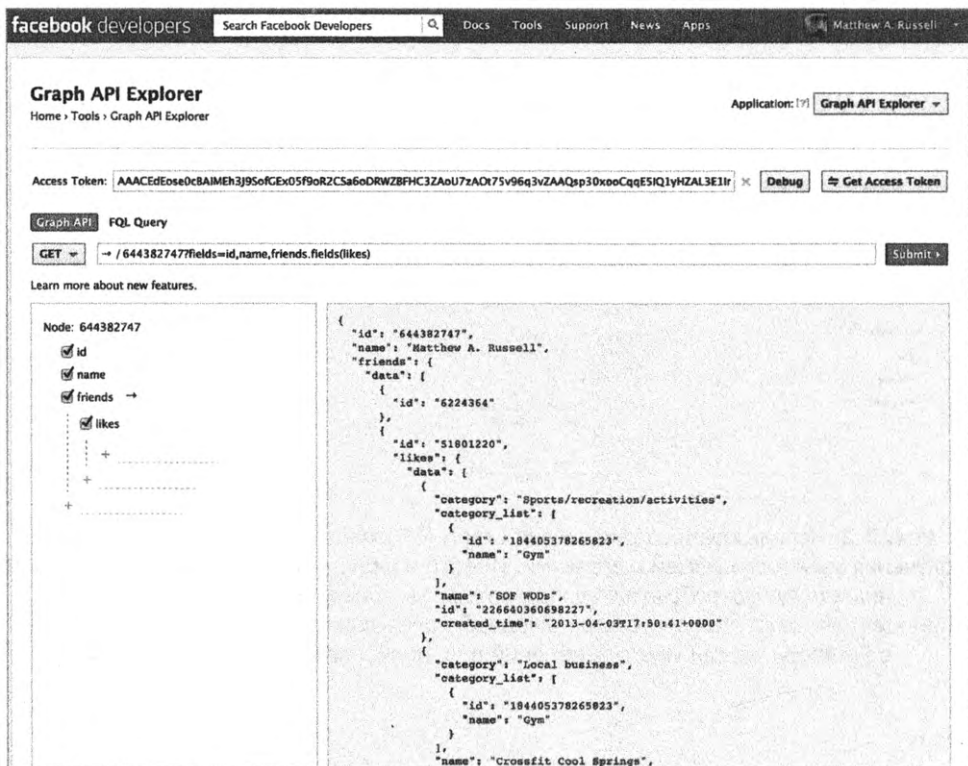


Рис. 2.3. Использование приложения Graph API Explorer для последовательного получения информации о заинтересованности друзей: запрашивается узел, связи с друзьями и «лайки» этих друзей. Этот пример наглядно показывает, насколько строже и сосредоточеннее на конфиденциальности пользователей стала политика доступа к данным в Facebook

Отладка

Кнопка Debug (Отладка) может пригодиться для поиска и устранения проблем в запросах, которые, по вашему мнению, должны возвращать данные, с учетом разрешений, связанных с токеном доступа, но ничего не возвращают.

Формат ответов JSON

В ответ на запросы Graph API возвращает результаты в удобном формате JSON, который легко поддается обработке.

Далее в этой главе мы будем исследовать Graph API программным способом, используя специализированный пакет на языке Python, однако есть возмож-

ность выполнять запросы к Graph API более непосредственно — через HTTP — например, симитировав запрос, который мы видели в Graph API Explorer. Код в примере 2.1 использует пакет `requests`¹ (вместо более громоздкого пакета из стандартной библиотеки Python, такого как `urllib`), чтобы упростить создание HTTP-запроса для получения списка ваших друзей и их лайков. Этот пакет можно установить, выполнив в терминале команду `pip install requests`. Запрос формируется на основе значений в параметре `fields` и идентичен запросу, который мы сконструировали в API Graph Explorer в интерактивном режиме. Особое внимание обратите на синтаксис `likes.limit(10){about}` — здесь используется механизм расширения полей² в Graph API, позволяющий выполнять вложенные подзапросы в одном вызове API.

Пример 2.1. Выполнение запросов к Graph API через HTTP

```
import requests # pip install requests
import json

base_url = 'https://graph.facebook.com/me'

# Определить извлекаемые поля
fields = 'id,name,likes.limit(10){about}'

url = '{0}?fields={1}&access_token={2}'.format(base_url, fields, ACCESS_TOKEN)

# Этот API доступен через HTTP, и запрос к нему можно отправить из браузера,
# с помощью утилиты командной строки, такой как curl, или из программы на
# любом языке, просто послав запрос по адресу URL.
# Чтобы убедиться в этом, щелкните по гиперссылке, которая появится в
# блокноте Jupiter Notebook после выполнения этого кода
print(url)

# Преобразовать ответ в формате JSON обратно в структуры
# данных языка Python
content = requests.get(url).json()

# Вывести содержимое ответа в отформатированном виде
print(json.dumps(content, indent=1))
```

Используя синтаксис расширения полей, в запросе к API можно устанавливать ограничения и смещения. Этот начальный пример просто показывает, что Facebook API реализован поверх HTTP. Ниже приводится несколько при-

¹ <http://bit.ly/1a1lrEt>.

² <http://bit.ly/1a1lsIE>.

меров использования ограничений/смещений, иллюстрирующих возможности селекторов полей:

```
# Получить 10 моих лайков
fields = 'id,name,likes.limit(10)'

# Получить следующие 10 моих лайков
fields = 'id,name,likes.offset(10).limit(10)'
```

Facebook API автоматически разбивает возвращаемые результаты на страницы, то есть если в ответ на ваш запрос будет найдено большое количество результатов, API не будет передавать их все сразу. Вместо этого результаты будут разбиты на фрагменты (*страницы*) и вы получите *курсор*, ссылающийся на следующую страницу с результатами. Дополнительные сведения о *разбиении результатов на страницы* смотрите в документации¹.

2.2.2. Знакомство с Open Graph Protocol

Вы должны знать, что кроме мощного программного интерфейса Graph API, который позволяет исследовать социальный граф и запрашивать известные объекты, платформа Facebook поддерживает протокол под названием Open Graph Protocol² (OGP), который был представлен еще в апреле 2010 года, на той же конференции F8, где был представлен социальный граф. OGP — это механизм, позволяющий разработчикам сделать любую веб-страницу объектом в социальном графе Facebook, внедрив в нее некоторые метаданные RDFa³. То есть кроме десятков объектов из «закрытой экосистемы» Facebook, описанных в справочнике⁴ Graph API (пользователи, фотографии, видео, отметки о посещении, ссылки, посты и т. д.), можно также встретить страницы из интернета, представляющие значимые понятия, которые были внедрены в социальный граф. Иначе говоря, OGP — это средство «расширения» социального графа, и в документации разработчика Facebook эта идея описывается в разделе «Open Graph»⁵.

¹ <http://bit.ly/1a1ltMP>.

² <http://bit.ly/1a1lu3m>.

³ <http://bit.ly/1a1lujR>.

⁴ <http://bit.ly/1a1lvEr>.

⁵ В этом разделе, описывающем реализацию OGP, термин *социальный граф* используется для обозначения обоих понятий — социального графа и открытого графа, если явно не оговаривается иное.

Возможности использования OGP для внедрения веб-страниц в социальный граф с разными целями практически ничем не ограничены, и, скорее всего, вы уже сталкивались с такими страницами, даже не осознавая этого. Например, взгляните на рис. 2.4, где показана страница фильма *The Rock* (Скала) на сайте IMDb.com¹. На боковой панели справа можно заметить довольно знакомую кнопку Like (Нравится) с сообщением «19,319 people like this. Be the first of your friends»². IMDb поддерживает эту функциональность, реализовав OGP для каждого из своих URL-адресов, соответствующих объектам, пригодным для включения в социальный граф. При наличии допустимых метаданных RDFa на странице Facebook сможет сформировать *связи* с этими объектами и включить их в потоки активности и другие ключевые элементы пользовательского интерфейса Facebook.

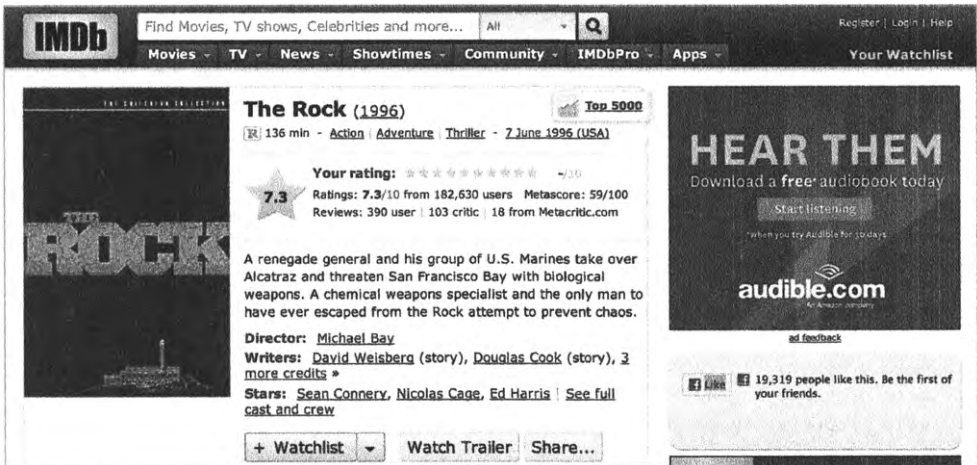


Рис. 2.4. Страница с описанием фильма «The Rock» на сайте IMDb, включающая поддержку OGP

Проявление реализации OGP в виде кнопки Like (Нравится) на веб-страницах может показаться немного банальным для привыкших видеть такие кнопки, однако тот факт, что в Facebook с успехом решили проблему расширения своей платформы, позволив произвольно включать в социальный граф объекты из интернета, имеет далекоидущие последствия.

¹ <http://imdb.com>.

² Понравился 19 319 пользователей. Станьте первым из ваших друзей. — *Примеч. пер.*

В 2013 году в Facebook был реализован механизм семантического поиска под названием Facebook Graph Search. Он дал пользователям возможность вводить в строку поиска запросы на естественном языке. Например, вы могли ввести в строку поиска запрос «мои друзья, которые живут в Лондоне и любят кошек» и получить список ваших друзей, которые действительно живут в Лондоне и любят кошек. Но этот новый способ выполнения запросов в Facebook продержался недолго. В конце 2014 года в Facebook отказались от поддержки семантического поиска, отдав предпочтение методу, основанному на ключевых словах. Однако граф, связывающий объекты Facebook с другими артефактами, никуда не делся. В ноябре 2017 года в Facebook выпустили автономное мобильное приложение Facebook Local, целью которого является включение в социальный граф физических мест и мероприятий. Например, приложение сообщит вам, какие рестораны пользуются популярностью у ваших друзей в Facebook. Основой этого приложения является социальный граф Facebook.

Давайте кратко рассмотрим суть реализации OGP, прежде чем перейти к запросам Graph API. Вот как выглядит канонический пример из документации OGP, показывающий, как превратить страницу IMDb фильма *The Rock* в объект с поддержкой протокола Open Graph, в виде документа XHTML, использующего пространства имен:

```
<html xmlns:og="http://ogp.me/ns#">
<head>
<title>The Rock (1996)</title>
<meta property="og:title" content="The Rock" />
<meta property="og:type" content="movie" />
<meta property="og:url" content="http://www.imdb.com/title/tt0117500/" />
<meta property="og:image" content="http://ia.media-imdb.com/images/rock.jpg" />
...
</head>
...
</html>
```

Эти метаданные обладают большим потенциалом при использовании в массовом масштабе, потому что позволяют идентификаторам URI, таким как <http://www.imdb.com/title/tt0117500>, однозначно представлять любые веб-страницы — людей, компаний, продуктов и т. д. — в машиночитаемом виде и продвигать их в семантической паутине. Кроме щелчка на кнопке Like (Нравится), чтобы выразить свое отношение к фильму *The Rock*, пользователи могут взаимодействовать с этим объектом другими способами. Например, пользователи могут сообщить, что они посмотрели¹ *The Rock*. OGP предлагает широкий и гибкий

¹ <http://bit.ly/2Qx0Kfo>.

набор взаимодействий между пользователями и объектами в рамках социального графа.



Если вы этого еще не сделали, загляните в исходный код HTML страницы <http://www.imdb.com/title/tt0117500> и посмотрите сами, как выглядит RDFa в дикой природе.

Запрос к Graph API для получения объекта Open Graph выглядит очень просто: добавьте URL веб-страницы или идентификатор объекта в `http(s)://graph.facebook.com/`, чтобы получить информацию об объекте. Например, введите в адресной строке браузера URL `http://graph.facebook.com/ http://www.imdb.com/title/tt0117500`, и в ответ вы получите:

```
{
  "share": {
    "comment_count": 0,
    "share_count": 1779
  },
  "og_object": {
    "id": "10150461355237868",
    "description": "Directed by Michael Bay. With Sean Connery, ...",
    "title": "The Rock (1996)",
    "type": "video.movie",
    "updated_time": "2018-09-18T08:39:39+0000"
  },
  "metadata": {
    "fields": [
      {
        "name": "id",
        "description": "The URL being queried",
        "type": "string"
      },
      {
        "name": "app_links",
        "description": "AppLinks data associated with the URL",
        "type": "applinks"
      },
      {
        "name": "development_instant_article",
        "description": "Instant Article object for the URL, in developmen...",
        "type": "instantarticle"
      },
      {
        "name": "instant_article",
        "description": "Instant Article object for the URL",

```

```

        "type": "instantarticle"
    },
    {
        "name": "og_object",
        "description": "Open Graph Object for the URL",
        "type": "opengraphobject:generic"
    },
    {
        "name": "ownership_permissions",
        "description": "Permissions based on ownership of the URL",
        "type": "urlownershippermissions"
    }
],
"type": "url"
},
"id": "http://www.imdb.com/title/tt0117500"
}

```

Заглянув в исходный код страницы <http://www.imdb.com/title/tt0117500>, вы обнаружите, что поля в ответе соответствуют данным в тегах `meta` страницы, и это не совпадение. Возврат богатого набора метаданных в ответ на простой запрос лишний раз доказывает, что OGP проектировался для работы. С помощью Graph API Explorer можно получить доступ к еще большему количеству метаданных, сопровождающих объекты в графе. Попробуйте вместо идентификатора объекта *The Rock* ввести в Graph API Explorer строку `380728101301?metadata=1`, чтобы запросить дополнительные метаданные. В ответ вы получите:

```

{
  "created_time": "2007-11-18T20:32:10+0000",
  "title": "The Rock (1996)",
  "type": "video.movie",
  "metadata": {
    "fields": [
      {
        "name": "id",
        "description": "The Open Graph object ID",
        "type": "numeric string"
      },
      {
        "name": "admins",
        "description": "A list of admins",
        "type": "list<opengraphobjectprofile>"
      },
      {
        "name": "application",
        "description": "The application that created this object",
        "type": "opengraphobjectprofile"
      }
    ]
  }
}

```

```
    },
    {
      "name": "audio",
      "description": "A list of audio URLs",
      "type": "list<opengraphobjectaudio>"
    },
    {
      "name": "context",
      "description": "Context",
      "type": "opengraphcontext"
    },
    {
      "name": "created_time",
      "description": "The time the object was created",
      "type": "datetime"
    },
    {
      "name": "data",
      "description": "Custom properties of the object",
      "type": "opengraphstruct:video.movie"
    },
    {
      "name": "description",
      "description": "A short description of the object",
      "type": "string"
    },
    {
      "name": "determiner",
      "description": "The word that appears before the object's title",
      "type": "string"
    },
    {
      "name": "engagement",
      "description": "The social sentence and like count for this object and its associated share. This is the same info used for the like button",
      "type": "engagement"
    },
    {
      "name": "image",
      "description": "A list of image URLs",
      "type": "list<opengraphobjectimagevideo>"
    },
    {
      "name": "is_scraped",
      "description": "Whether the object has been scraped",
      "type": "bool"
    },
    {
      "name": "locale",
```



```

    "description": "The locale the object is in",
    "type": "opengraphobjectlocale"
  },
  {
    "name": "location",
    "description": "The location inherited from Place",
    "type": "location"
  },
  {
    "name": "post_action_id",
    "description": "The action ID that created this object",
    "type": "id"
  },
  {
    "name": "profile_id",
    "description": "The Facebook ID of a user that can be followed",
    "type": "opengraphobjectprofile"
  },
  {
    "name": "restrictions",
    "description": "Any restrictions that are placed on this object",
    "type": "opengraphobjectrestrictions"
  },
  {
    "name": "see_also",
    "description": "An array of URLs of related resources",
    "type": "list<string>"
  },
  {
    "name": "site_name",
    "description": "The name of the web site upon which the object resides",
    "type": "string"
  },
  {
    "name": "title",
    "description": "The title of the object as it should appear
                  in the graph",
    "type": "string"
  },
  {
    "name": "type",
    "description": "The type of the object",
    "type": "string"
  },
  {
    "name": "updated_time",
    "description": "The last time the object was updated",
    "type": "datetime"
  },
},

```

```

{
  "name": "video",
  "description": "A list of video URLs",
  "type": "list<opengraphobjectimagevideo>"
}
],
"type": "opengraphobject:video.movie",
"connections": {
  "comments": "https://graph.facebook.com/v3.1/380728101301/comments?access_token=EAAyvPRk4YUEBAHvVKqnhZBxMDAwBKEpWrsM6J8ZCxCxHkLu...&pretty=0",
  "likes": "https://graph.facebook.com/v3.1/380728101301...&pretty=0",
  "picture": "https://graph.facebook.com/v3.1/380728101...&pretty=0",
  "reactions": "https://graph.facebook.com/v3.1/38072810...reactions?access_token=EAAyvPRk4YUEBAHvVKqnhZBxMDAwBKEpWrsM6J8ZC...&pretty=0"
}
},
"id": "380728101301"
}

```

Элементы в `metadata.connections` — это ссылки на другие узлы в графе, которые можно запросить и получить другую интригующую информацию, но имейте в виду, что ваши попытки могут быть сильно ограничены настройками конфиденциальности в Facebook.



Попробуйте использовать Facebook-идентификатор «MiningTheSocialWeb» для получения сведений об официальной фан-странице этой книги¹ в Graph API Explorer. Также можете попробовать изменить пример 2.1 и выполнить запрос для `https://graph.facebook.com/MiningTheSocialWeb` программно, чтобы получить основную информацию о странице, включая ее содержимое.

Например, добавив в этот URL строку запроса `"?fields=posts"`, вы получите опубликованные на ней сообщения.

И в заключение, перед тем как перейти к исследованию программных методов доступа к Graph API, призываем вас проявлять творческую смекалку, изучая возможности OGP, но имейте в виду, что этот протокол все еще продолжает развиваться. Поскольку речь идет о семантической паутине и веб-стандартах в целом, использование термина «открытый»² по понятным причинам вызвало некоторое недоумение. За последние годы в спецификацию было добавлено много нового³, и что-то, вероятно, еще будет добавлено. Также имейте в виду, что OGP, по сути, является плодом усилий единственного производителя,

¹ <http://on.fb.me/1a1AI8>.

² <http://tcn.ch/1a1IAYF>.

³ <http://bit.ly/1a1Abd>.

который по своим масштабам ничуть не уступает внедрению элементов *meta*¹ на ранних стадиях развития Всемирной паутины, хотя социальные эффекты, похоже, ведут к совсем другому результату.

Будет ли OGP или какой-то преемник Graph Search доминировать в интернете в будущем — спорный вопрос, но потенциал определенно имеется; индикаторы успеха неуклонно изменяются в положительном направлении, и много интересного может произойти по мере появления инноваций в будущем. Теперь, когда вы получили более полное представление о социальном графе, обернемся и посмотрим, как получить доступ к Graph API.

2.3. Анализ связей в социальном графе

Официальный пакет SDK на Python для работы с Graph API², первоначально созданный в Facebook и ныне поддерживаемый сообществом, можно установить стандартным способом с помощью `pip`, выполнив команду **`pip install facebook-sdk`**. Этот пакет содержит несколько удобных методов для взаимодействия с Facebook разными способами. Но на самом деле класс `GraphAPI` (определен в файле `facebook.py`) содержит всего несколько ключевых методов, о которых вы должны знать, чтобы извлекать данные из Graph API, поэтому при желании вы можете выполнять HTTP-запросы непосредственно с помощью модуля `requests` (как было показано в примере 2.1). Вот эти методы:

```
get_object(self, id, **args)
```

Пример использования: `get_object("me", metadata=1)`

```
get_objects(self, id, **args)
```

Пример использования: `get_objects(["me", "some_other_id"], metadata=1)`

```
get_connections(self, id, connection_name, **args)
```

Пример использования: `get_connections("me", "friends", metadata=1)`

```
request(self, path, args=None, post_args=None)
```

Пример использования: `request("search", {"q": "social web", "type": "page"})`

¹ <http://bit.ly/1a1lMa>.

² <http://bit.ly/2kpej52>.



Ограничения на количество запросов к API, установленные в Facebook, зависят от числа пользователей. Чем больше людей будет пользоваться вашим приложением, тем меньше ограничений будет накладываться. Тем не менее тщательно прорабатывайте свое приложение, чтобы как можно меньше использовать API, и старайтесь обрабатывать все ошибки.

Наиболее распространенным (и часто единственным) именованным аргументом, который вы будете использовать для получения связей объекта, кроме информации о нем самом, является `metadata=1`. Взгляните на пример 2.2, который представляет класс `GraphAPI` и использует его методы для получения информации о вас, ваших связях и для отправки *поискового запроса*, такого как *social web*. В этом примере также представлена вспомогательная функция `pp`, которая будет использоваться в оставшейся части этой главы для форматированного вывода результатов в JSON, чтобы избавить вас от лишнего ввода.



Программный интерфейс Facebook претерпел некоторые изменения, и теперь для извлечения общедоступного контента со страниц Facebook программным способом требуется получить дополнительные привилегии, отправив приложение на проверку и утверждение¹.

Пример 2.2. Вызов Graph API из Python

```
import facebook # pip install facebook-sdk
import json

# Вспомогательная функция для форматированного вывода объектов Python
# в формате JSON
def pp(o):
    print(json.dumps(o, indent=1))

# Подключиться к Graph API с использованием токена доступа
g = facebook.GraphAPI(ACCESS_TOKEN, version='2.7')

# Выполнить несколько запросов:

# Получить свой идентификатор
pp(g.get_object('me'))

# Получить связи для этого идентификатора
# Примеры названий связей: 'feed', 'likes', 'groups', 'posts'
pp(g.get_connections(id='me', connection_name='likes'))
```

¹ <http://bit.ly/2vDb2B1>.

```
# Для поиска местоположений может потребоваться получить одобрение
# приложения в Facebook
pp(g.request("search", {'type': 'place', 'center': '40.749444, -73.968056',
                        'fields': 'name, location'}))
```

Запрос, выполняющий поиск местоположения, интересен тем, что возвращает элементы графа, географически близкие к указанной широте и долготе. Вот несколько примеров ответов на этот запрос:

```
{
  "data": [
    {
      "name": "United Nations",
      "location": {
        "city": "New York",
        "country": "United States",
        "latitude": 40.748801288774,
        "longitude": -73.968307971954,
        "state": "NY",
        "street": "United Nations Headquarters",
        "zip": "10017"
      },
      "id": "54779960819"
    },
    {
      "name": "United Nations Security Council",
      "location": {
        "city": "New York",
        "country": "United States",
        "latitude": 40.749283619093,
        "longitude": -73.968088677538,
        "state": "NY",
        "street": "760 United Nations Plaza",
        "zip": "10017"
      },
      "id": "113874638768433"
    },
    {
      "name": "New-York, Time Square",
      "location": {
        "city": "New York",
        "country": "United States",
        "latitude": 40.7515,
        "longitude": -73.97076,
        "state": "NY"
      },
      "id": "1900405660240200"
    }
  ]
}
```

```

    },
    {
      "name": "Penn Station, Manhattan, New York",
      "location": {
        "city": "New York",
        "country": "United States",
        "latitude": 40.7499131,
        "longitude": -73.9719497,
        "state": "NY",
        "zip": "10017"
      },
      "id": "1189802214427559"
    },
    {
      "name": "Central Park Manhattan",
      "location": {
        "city": "New York",
        "country": "United States",
        "latitude": 40.7660016,
        "longitude": -73.9765709,
        "state": "NY",
        "zip": "10021"
      },
      "id": "328974237465693"
    },
    {
      "name": "Delegates Lounge, United Nations",
      "location": {
        "city": "New York",
        "country": "United States",
        "latitude": 40.749433,
        "longitude": -73.966938,
        "state": "NY",
        "street": "UN Headquarters, 10017",
        "zip": "10017"
      },
      "id": "198970573596872"
    },
    ...
  ],
  "paging": {
    "cursors": {
      "after": "MjQZD"
    },
    "next": "https://graph.facebook.com/v2.5/search?access_token=..."
  }
}

```



Используя Graph API Explorer, вы получили бы идентичные результаты. Во время разработки часто бывает очень удобно использовать Graph API Explorer и Jupyter Notebook в тандеме, в зависимости от конкретной цели. Преимущество Graph API Explorer заключается в той легкости, с какой можно поощрять новые запросы, щелкая мышью на значениях идентификаторов.

Сейчас в ваших руках сосредоточена вся мощь Graph API Explorer и консоли Python и все, что они могут предложить. И теперь, заглянув в экосистему, давайте попробуем проанализировать некоторые ее данные.

2.3.1. Анализ страниц в Facebook

Проект Facebook начинался как сайт чистой социальной сети без социального графа и не предоставлявший бизнесу способа присутствия в Сети. Однако быстро адаптировался под потребности рынка. Теперь предприятия, клубы, книги и многие другие неодушевленные сущности имеют страницы Facebook¹ и своих поклонников. Страницы Facebook — мощный инструмент для бизнеса, помогающий привлекать клиентов, и, осознавая это, в Facebook предприняли некоторые меры, предложив администраторам страниц Facebook небольшой набор инструментов для изучения их аудитории, который получил говорящее название Insights².

Если у вас есть аккаунт в Facebook, вполне возможно, что вам уже понравилась одна или несколько страниц, которые вы одобряете или считаете интересными, и в этом смысле страницы Facebook значительно расширяют возможности социального графа как платформы. Явная возможность размещения страниц Facebook для неодушевленных сущностей, кнопка Like (Нравится) и ткань социального графа в совокупности обеспечивают мощный арсенал для платформы графа интересов, которая несет множество возможностей. (Подробно о том, почему графы интересов настолько богаты полезными возможностями, рассказывается в разделе «Причины популярности Twitter» в главе 1.)

Анализ страницы Facebook этой книги

Учитывая, что для этой книги имеется соответствующая страница Facebook, которая оказывается в верхних строчках результатов при поиске по фразе

¹ <http://on.fb.me/1a1lCzQ>.

² Название *Insights* в данном случае можно перевести как *познание, анализ*. — *Примеч. пер.*

«social web», кажется вполне естественным использовать ее здесь, в этой главе, в качестве отправной точки для иллюстрации некоторых видов анализа.¹

Вот лишь несколько вопросов в отношении страницы этой книги в Facebook, которые, вероятно, стоит рассмотреть:

- Насколько популярна страница?
- Насколько велика вовлеченность поклонников страницы?
- Есть ли среди поклонников особенно яркие и активные?
- Какие темы на странице обсуждаются наиболее активно?

Круг вопросов, которые можно задать Graph API при исследовании страницы Facebook, ограничивается только вашим воображением, и вопросы, перечисленные выше, помогут вам выбрать правильное направление. Кроме того, эти вопросы мы также будем использовать как основу для сравнения с другими страницами.

Итак, мы могли бы начать наше путешествие с поискового запроса «social web», в ответ на который среди прочих возвращается книга с названием *Mining the Social Web* в виде следующего элемента:

```
{
  "data": [
    {
      "name": "Mining the Social Web",
      "id": "146803958708175"
    },
    {
      "name": "R: Mining spatial, text, web, and social media",
      "id": "321086594970335"
    }
  ],
  "paging": {
    "cursors": {
      "before": "MAZDZD",
      "after": "MQZDZD"
    }
  }
}
```

¹ Читая этот раздел, не забывайте, что Facebook ограничивает доступ к публичному контенту для приложений, которые не были одобрены компанией. Код в этом разделе и производимые им данные приводятся исключительно для иллюстрации. За более подробной информацией обращайтесь к документации разработчика: <https://developers.facebook.com/docs/apps/review>.

Мы можем взять идентификатор любого из элементов в результатах поиска и использовать его как основу запроса к графу, который можно послать вызовом метода `get_object` экземпляра `facebook.GraphAPI`. В отсутствие числового идентификатора можно выполнить поисковый запрос по названию и посмотреть, что он вернет. С помощью метода `get_object` можно получить дополнительную информацию, например количество поклонников страницы Facebook, как показано в примере 2.3.

Пример 2.3. Отправка запросов в Graph API для получения идентификатора «Mining the Social Web» и количества поклонников книги

```
# Найти идентификатор книги по названию
pp(g.request("search", {'q': 'Mining the Social Web', 'type': 'page'}))

# Извлечь идентификатор и запросить количество поклонников
mtsw_id = '146803958708175'
pp(g.get_object(id=mtsw_id, fields=['fan_count']))
```

Этот код вернет следующие результаты:

```
{
  "data": [
    {
      "name": "Mining the Social Web",
      "id": "146803958708175"
    },
    {
      "name": "R: Mining spatial, text, web, and social media",
      "id": "321086594970335"
    }
  ],
  "paging": {
    "cursors": {
      "before": "MAZDZD",
      "after": "MQZDZD"
    }
  }
}
{
  "fan_count": 2563,
  "id": "146803958708175"
}
```

Определение количества поклонников страницы и сравнение его с аналогичным показателем других страниц из той же категории — это способ измерения силы «бренда» в Facebook. *Mining The Social Web* — довольно узкоспециализированная техническая книга, поэтому имеет смысл сравнить ее с другими книгами,

опубликованными издательством O'Reilly Media и имеющими свои страницы Facebook.

Для любого анализа популярности и понимания более широкого контекста необходимы сопоставимые данные. Провести сравнение можно разными способами, но есть несколько интересных моментов: на момент написания этой книги страница издательства O'Reilly Media¹ имела около 126 000 лайков, а страница языка программирования Python² — около 121 000. То есть число поклонников *Mining the Social Web* составляет около 2% от числа поклонников издательства и языка Python. Очевидно, что есть определенный потенциал для роста популярности этой книги, даже притом, что она является узкоспециализированной.

Конечно, лучше было бы провести сравнение с такой же специализированной книгой, как *Mining the Social Web*, но найти хороших кандидатов для такого сравнения, просматривая данные страниц Facebook, очень непросто. Например, нет никакой возможности ограничить поиск только страницами книг; вместо этого вам придется отыскать как можно больше страниц, а затем отфильтровать набор результатов по категориям, оставив только книги. И все же эта задача имеет несколько вариантов решения.

Один из них — отыскать книги с похожими названиями, изданные в O'Reilly. Например, на момент написания второго издания этой книги поиск в Graph API по запросу *Programming Collective Intelligence* (похожая книга, изданная в O'Reilly) возвращал страницу сообщества с примерно 925 лайками.

Другой вариант — использовать для сравнения возможности протокола Facebook Open Graph. Например, онлайн-каталог O'Reilly включает поддержку OGP для всех книг, перечисленных в нем, имеются кнопки Like (Нравится) для обеих книг, *Mining the Social Web*, 2nd Edition³ и *Programming Collective Intelligence*⁴. Мы легко можем выполнить запросы к Graph API и посмотреть, какие данные доступны, просто запрашивая эти URL-адреса в браузере:

Запрос к Graph API для Mining the Social Web

<https://graph.facebook.com/http://shop.oreilly.com/product/0636920030195.do>

¹ <http://on.fb.me/1a1ID6F>.

² <http://on.fb.me/1a1ID6V>.

³ <http://oriel.ly/1cMLoug>.

⁴ <http://oriel.ly/1a1IGzw>.

Запрос к Graph API для Programming Collective Intelligence

<https://graph.facebook.com/http://shop.oreilly.com/product/9780596529321.do>

С точки зрения выполнения запросов на Python программным способом URL-адреса — это запрашиваемые объекты (в точности как URL-адрес записи в IMDb для фильма *The Rock* в примере выше), то есть мы можем запрашивать эти объекты, как показано в примере 2.4.

Пример 2.4. Запросы к Graph API для получения объектов Open Graph по их адресам URL

```
# Ссылка на запись в каталоге для книги MTSW
pp(g.get_object('http://shop.oreilly.com/product/0636920030195.do'))

# Ссылка на запись в каталоге для книги PCI
pp(g.get_object('http://shop.oreilly.com/product/9780596529321.do'))
```

Имейте в виду тонкое, но очень важное отличие: даже притом, что страница в каталоге O'Reilly и фан-страница Facebook для *Mining the Social Web* логически представляют одну и ту же книгу, узлы (и сопутствующие метаданные, такие как количество лайков), соответствующие этим страницам, совершенно независимы. Это всего лишь совпадение, что каждый из них представляет один и тот же реальный предмет.



Существует совершенно отдельный вид анализа, известный как *разрешение сущностей* (или *устранение неоднозначности сущностей*, в зависимости от формулировки задачи), помогающий объединить упоминания предметов в одно номинальное понятие. Например, в данном случае процесс разрешения сущностей мог бы заметить, что в Open Graph имеется несколько узлов, в действительности ссылающихся на одно и то же номинальное понятие *Mining the Social Web*, и создать связи между ними, указав, что в действительности они представляют одну и ту же сущность в реальном мире. Разрешение сущностей — захватывающая область исследований, которая в будущем окажет глубокое влияние на наши подходы к использованию данных.

На рис. 2.5 показан сеанс исследования Graph API с помощью Jupyter Notebook.

Однако на практике нечасто удастся найти подходящего кандидата для сравнения и получить результат, заслуживающий доверия, и приходится продолжать исследования. Изучение набора данных, достаточно большого, чтобы накопить знания о них, часто обеспечивает всю необходимую информацию, которая вам понадобится, когда вы впервые окажетесь в пространстве задачи.

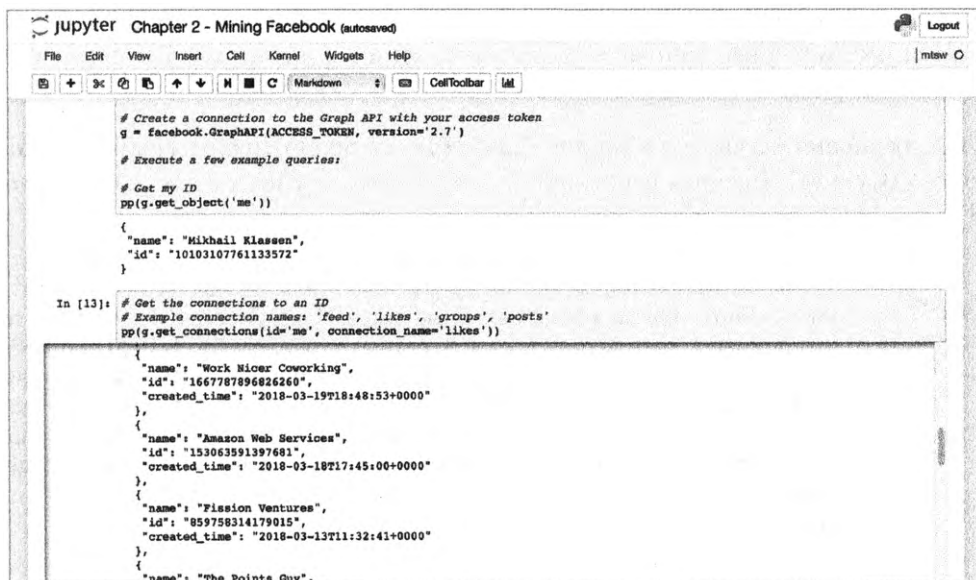


Рис. 2.5. Исследование Graph API с особенной легкостью можно выполнять, пользуясь интерактивной средой программирования, такой как Jupyter Notebook

Вовлеченность поклонников и измерение силы бренда в социальной сети

Некоторые знаменитости проявляют высокую активность в социальных сетях, другие оставляют задачу привлечения онлайн-аудитории специальной группе по маркетингу в социальных сетях или команде по связям с общественностью. Страницы Facebook предлагают отличный способ привлечения поклонников, кем бы вы ни были: крупной знаменитостью, начинающей звездой YouTube или просто поддерживаете в Facebook страницу некоммерческой организации, в работе которой вы принимаете участие на добровольных началах.

В этом разделе мы рассмотрим примеры анализа страниц Facebook трех очень популярных музыкантов, чтобы узнать, насколько сильно их аудитория в Facebook реагирует на то, что они публикуют на своих страницах. Вам станет понятно, почему эти артисты (и их менеджеры по рекламе) с особым вниманием относятся к этой информации. Привлечение и удержание поклонников жизненно важно для их успеха, продажи билетов на мероприятия или мобилизации поклонников с некоторой целью.

Эффективное общение с аудиторией имеет большое значение, и, допустив ошибку в посте, вы можете оттолкнуть некоторых из ваших самых преданных

поклонников. По этой причине артисты и другие публичные личности стремятся иметь точные данные о вовлеченности их аудитории в Facebook. Вот о чем этот раздел.

Для сравнения мы выбрали Тейлор Свифт (Taylor Swift), Дрейка (Drake) и Бейонсе (Beyoncé). Все они необычайно талантливы, и у всех их есть страницы Facebook с большим числом фолловеров.



Чтобы найти идентификатор каждого исполнителя, вы можете использовать инструменты поиска, представленные выше. И, разумеется, несмотря на то что примеры ниже используют конкретные страницы Facebook, сами идеи являются общими. Вы можете попробовать получить аналогичную статистику на других платформах, таких как Twitter. Или написать код, который будет сканировать новостные порталы в поисках упоминаний определенной знаменитости или бренда. Если вы когда-либо использовали оповещения Google (Google Alerts), значит, вы должны иметь представление о том, как это может выглядеть.

В примере 2.5 для каждого исполнителя задается идентификатор его страницы, который можно получить, выполнив поиск, как показано в примере 2.3. Затем определяется вспомогательная функция для получения числа поклонников в виде целого числа. Эти числа сохраняются в трех переменных, а затем выводятся на экран.



На момент написания этих строк, чтобы получить контент публичной страницы из Facebook, приложение требовалось представить на проверку¹ и получить одобрение. Это результат усилий Facebook по повышению безопасности своей платформы и предотвращению злоупотреблений.

Пример 2.5. Подсчет числа поклонников страницы

```
# Для правильной работы следующего кода разработчику приложения
# может потребоваться передать его для проверки и утверждения.
# См. https://developers.facebook.com/docs/apps/review.

# Возьмем для примера страницы трех популярных музыкантов
taylor_swift_id = '19614945368'
drake_id = '83711079303'
beyonce_id = '28940545600'

# Вспомогательная функция, получающая число поклонников ('likes')
```

¹ <http://bit.ly/2vDb2B1>.

```
# страницы
def get_total_fans(page_id):
    return int(g.get_object(id=page_id, fields=['fan_count'])['fan_count'])

tswift_fans = get_total_fans(taylor_swift_id)
drake_fans = get_total_fans(drake_id)
beyonce_fans = get_total_fans(beyonce_id)

print('Taylor Swift: {0} fans on Facebook'.format(tswift_fans))
print('Drake: {0} fans on Facebook'.format(drake_fans))
print('Beyoncé: {0} fans on Facebook'.format(beyonce_fans))
```

Если запустить этот код, он выведет примерно следующие результаты:

```
Taylor Swift: 73896104 fans on Facebook
Drake: 35821534 fans on Facebook
Beyoncé: 63974894 fans on Facebook
```

Общее количество поклонников в Facebook — первый и самый основной показатель популярности страницы. Но мы пока не знаем, насколько активны эти поклонники, какой наиболее вероятной будет их реакция на пост и будут ли они принимать активное участие, комментируя посты или делаясь ими с другими. Все эти разные реакции важны для владельца страницы, потому что у каждого из поклонников может быть много друзей в Facebook, и алгоритм ленты новостей в Facebook часто сообщает об активности пользователей их друзьям. Активное участие поклонников приведет к тому, что посты увидят больше людей, к ним будет привлечено больше внимания, а значит, будет больше поклонников, больше лайков, больше продаж и всего остального, чем вы пытаетесь управлять.

Чтобы измерить уровень вовлеченности, необходимо получить ленту постов страницы. А для этого нужно подключиться к Graph API, используя идентификатор страницы, и выбрать посты. Записи будут возвращены в виде объектов JSON с большим количеством метаданных. Python интерпретирует JSON как словарь с ключами и значениями.

В примере 2.6 сначала определяется функция, которая извлекает ленту сообщений страницы, перемещает данные из ленты в список и возвращает заданное количество записей. Поскольку API автоматически разбивает результаты на страницы, функция продолжит запрашивать страницы, пока не будет получено заданное количество записей, а затем вернет их. Далее, получив список записей из ленты страницы, нужно извлечь информацию из этих записей. Нам может быть интересно увидеть текст поста — например, адресованного поклонни-

кам, — поэтому следующая вспомогательная функция извлекает и возвращает текст сообщения.

Пример 2.6. Извлечение ленты постов из страницы

Вспомогательная функция для извлечения ленты постов из указанной страницы

```
def retrieve_page_feed(page_id, n_posts):  
    """Извлекает первые n_posts сообщений из ленты  
    в обратном хронологическом порядке."""  
    feed = g.get_connections(page_id, 'posts')  
    posts = []  
    posts.extend(feed['data'])  
  
    while len(posts) < n_posts:  
        try:  
            feed = requests.get(feed['paging']['next']).json()  
            posts.extend(feed['data'])  
        except KeyError:  
            # Если сообщений в ленте больше нет, прервать цикл  
            print('Reached end of feed.')  
            break  
  
    if len(posts) > n_posts:  
        posts = posts[:n_posts]  
  
    print('{} items retrieved from feed'.format(len(posts)))  
    return posts
```

Вспомогательная функция для извлечения текста из поста

```
def get_post_message(post):  
    try:  
        message = post['story']  
    except KeyError:  
        # Сообщение может иметь поле 'message' вместо 'story'  
        pass  
    try:  
        message = post['message']  
    except KeyError:  
        # Сообщение не содержит текста  
        message = ''  
    return message.replace('\n', ' ')
```

Извлечь 5 последних элементов из ленты

```
for artist in [taylor_swift_id, drake_id, beyonce_id]:  
    print()  
    feed = retrieve_page_feed(artist, 5)  
    for i, post in enumerate(feed):  
        message = get_post_message(post)[:50]  
        print('{} - {}'.format(i+1, message))
```

Цикл в заключительном блоке кода перебирает трех исполнителей в нашем примере, получает последние 5 записей в их лентах и выводит на экран первые 50 символов из каждого поста. Вот что получилось в результате:

```
5 items retrieved from feed
1 - Check out a key moment in Taylor writing "This Is ...
2 - ...
3 - ...
4 - The Swift Life is available for free worldwide in ...
5 - #TheSwiftLife App is available NOW for free in the...
```

```
5 items retrieved from feed
1 - ...
2 - http://www.hollywoodreporter.com/features/drakes-h...
3 - ...
4 - Tickets On Sale Friday, September 15....
5 - https://www.youcaring.com/jjwatt...
```

```
5 items retrieved from feed
1 - ...
2 - ...
3 - ...
4 - New Shop Beyoncé 2017 Holiday Capsule: shop.beyonc...
5 - Happy ThiccsGiving. www.beyonce.com...
```

Элементы, включающие только многоточие (...), не содержали текста (не забывайте, что Facebook позволяет также публиковать в ленте видео и фотографии).

Facebook предлагает своим пользователям много разных способов взаимодействия с постом. Кнопка Like (Нравится) впервые появилась в 2009 году и была заимствована всеми социальными сетями. Поскольку Facebook и многие другие платформы социальных сетей используются рекламодателями, Like или что-то подобное ей посылает сильный сигнал о том, что пост резонирует с помыслами целевой аудитории. Это привело к неожиданным последствиям: многие все чаще стараются писать посты так, чтобы получить больше лайков.

В 2016 году Facebook расширил возможные реакции, добавив кнопки «Love» (Супер), «Haha» (Ха-ха), «Wow» (Ух ты!), «Sad» (Сочувствую) и «Angry» (Возмутительно). По состоянию на май 2017 года пользователи также могут выражать свое отношение к комментариям.

Кроме выражения эмоциональной реакции, пользователи могут комментировать посты или делиться ими. Любое из этих действий увеличивает вероятность появления исходного поста в лентах новостей других пользователей. Хотелось бы иметь инструменты, с помощью которых можно узнать, сколько

пользователей откликнулось на пост любым из этих способов. Для простоты ограничимся только кнопкой Like (Нравится), хотя код можно изменить и исследовать ответы с большей детализацией.

Количество откликов на пост будет примерно пропорционально численности аудитории этого поста. Количество поклонников на странице — это один из показателей размера аудитории, но нужно учитывать, что не все поклонники обязательно увидят каждый пост. Кто и что увидит, определяется алгоритмом лент новостей в Facebook, кроме того, авторы постов могут дополнительно «расширить» круг тех, кто увидит пост, заплатив Facebook рекламный сбор.

Предположим, что в нашем примере все поклонники каждого артиста видят каждый пост, и попробуем узнать, какая доля поклонников выразила свое отношение к сообщению, лайкнув его, оставив комментарий или расшарив его. Это позволит сравнить сообщества поклонников: у кого самые активные поклонники; чьими постами делятся чаще всего.

Агентам артистов, стремящимся добиться максимального интереса к каждому посту, может быть интересно узнать, какие посты вызывают наибольший резонанс. На какой контент активнее реагируют поклонники — видео, фотографии или текст? Меняется ли активность поклонников в зависимости от дня недели или времени суток? Последний вопрос о времени публикации поста не имеет большого значения для Facebook, потому что алгоритм ленты новостей контролирует, кто и что увидит, тем не менее эта информация может быть полезна.

В примере 2.7 мы объединяем разные фрагменты в общую картину. Вспомогательная функция `measure_response` подсчитывает количество лайков, комментариев и ссылок, собирает все посты. Другая вспомогательная функция, `measure_engagement`, соотносит эти числа с общим количеством поклонников.

Пример 2.7. Измерение вовлеченности

```
# Определяет число откликов на пост -- лайков, комментариев и ссылок
def measure_response(post_id):
    """Возвращает число реакций на заданный пост,
    которое может служить мерой вовлеченности пользователей."""
    likes = g.get_object(id=post_id,
                        fields=['likes.limit(0).summary(true)'])\
            ['likes']['summary']['total_count']
    shares = g.get_object(id=post_id,
                        fields=['shares.limit(0).summary(true)'])\
            ['shares']['count']
```

```

comments = g.get_object(id=post_id,
                        fields=['comments.limit(0).summary(true)'])\
                        ['comments']['summary']['total_count']
return likes, shares, comments

# Определяет относительную активность поклонников, вызванную
# публикацией заданного поста
def measure_engagement(post_id, total_fans):
    """Возвращает процентную долю поклонников,
    среагировавших на заданный пост."""
    likes = g.get_object(id=post_id,
                        fields=['likes.limit(0).summary(true)'])\
                        ['likes']['summary']['total_count']
    shares = g.get_object(id=post_id,
                        fields=['shares.limit(0).summary(true)'])\
                        ['shares']['count']
    comments = g.get_object(id=post_id,
                        fields=['comments.limit(0).summary(true)'])\
                        ['comments']['summary']['total_count']
    likes_pct = likes / total_fans * 100.0
    shares_pct = shares / total_fans * 100.0
    comments_pct = comments / total_fans * 100.0
    return likes_pct, shares_pct, comments_pct

# Извлечь последние 5 элементов из лент сообщений артистов и вывести
# реакции на них и уровень вовлеченности
artist_dict = {'Taylor Swift': taylor_swift_id,
               'Drake': drake_id,
               'Beyoncé': beyonce_id}

for name, page_id in artist_dict.items():
    print()
    print(name)
    print('-----')
    feed = retrieve_page_feed(page_id, 5)
    total_fans = get_total_fans(page_id)
    for i, post in enumerate(feed):
        message = get_post_message(post)[:30]
        post_id = post['id']
        likes, shares, comments = measure_response(post_id)
        likes_pct, shares_pct, comments_pct = measure_engagement(post_id,
                                                                total_fans)
        print('{0} - {1}...'.format(i+1, message))
        print('    Likes {0} ({1:7.5f}%)' .format(likes, likes_pct))
        print('    Shares {0} ({1:7.5f}%)' .format(shares, shares_pct))
        print('    Comments {0} ({1:7.5f}%)' .format(comments, comments_pct))

```

Цикл по трем нашим артистам и измерение вовлеченности дали следующие результаты:

Taylor Swift

5 items retrieved from feed

1 - Check out a key moment in Tayl...

Likes 33134 (0.04486%)

Shares 1993 (0.00270%)

Comments 1373 (0.00186%)

2 - ...

Likes 8282 (0.01121%)

Shares 19 (0.00003%)

Comments 353 (0.00048%)

3 - ...

Likes 11083 (0.01500%)

Shares 8 (0.00001%)

Comments 383 (0.00052%)

4 - The Swift Life is available fo...

Likes 39237 (0.05312%)

Shares 926 (0.00125%)

Comments 1012 (0.00137%)

5 - #TheSwiftLife App is available...

Likes 60721 (0.08221%)

Shares 1895 (0.00257%)

Comments 2105 (0.00285%)

Drake

5 items retrieved from feed

1 - ...

Likes 23938 (0.06685%)

Shares 2907 (0.00812%)

Comments 3785 (0.01057%)

2 - <http://www.hollywoodreporter.c...>

Likes 4474 (0.01250%)

Shares 166 (0.00046%)

Comments 310 (0.00087%)

3 - ...

Likes 44887 (0.12536%)

Shares 8 (0.00002%)

Comments 1895 (0.00529%)

4 - Tickets On Sale Friday, Septem...

Likes 19003 (0.05307%)

Shares 1343 (0.00375%)

Comments 6459 (0.01804%)

5 - <https://www.youcaring.com/jjwa...>

Likes 17109 (0.04778%)

Shares 1777 (0.00496%)

Comments 859 (0.00240%)

```
Beyoncé
-----
5 items retrieved from feed
1 - ...
    Likes 8328 (0.01303%)
    Shares 134 (0.00021%)
    Comments 296 (0.00046%)
2 - ...
    Likes 18545 (0.02901%)
    Shares 250 (0.00039%)
    Comments 819 (0.00128%)
3 - ...
    Likes 21589 (0.03377%)
    Shares 460 (0.00072%)
    Comments 453 (0.00071%)
4 - New Shop Beyoncé 2017 Holiday ...
    Likes 10717 (0.01676%)
    Shares 246 (0.00038%)
    Comments 376 (0.00059%)
5 - Happy Thiccgiving. www.beyonc...
    Likes 25497 (0.03988%)
    Shares 653 (0.00102%)
    Comments 610 (0.00095%)
```

Вовлеченность всего 0,04% от общего числа поклонников может показаться низкой, но не забывайте, что большинство людей никак не реагируют на большинство постов, которые они видят. И когда у вас десятки миллионов поклонников, мобилизация даже небольшой их части может иметь большое влияние.

2.3.2. Манипулирование данными с помощью pandas

Библиотека `pandas` для Python — важный инструмент в арсенале любого исследователя данных, и мы будем использовать ее далее в этой книге. Она предлагает высокопроизводительные структуры данных для хранения табличных данных и мощные средства анализа, написанные на Python, при этом некоторые части библиотеки, выполняющие особенно интенсивные вычисления, написаны на C или Cython. Разработка библиотеки была начата в 2008 году Уэсом Маккинни (Wes McKinney), и первоначально она предназначалась для анализа финансовых данных.

Одной из основных структур данных, которые предлагает `pandas`, является `DataFrame`, которая по сути похожа на базу данных или таблицу. Она имеет столбцы с метками и индекс. Структура эффективно обрабатывает недостающие

данные, поддерживает временные ряды и индексирование по времени, позволяет объединять данные и получать срезы, а также имеет множество удобных инструментов для чтения и записи данных.

Узнать больше о библиотеке можно в репозитории `pandas` на сайте GitHub¹ и в официальной документации². Краткое введение в библиотеку вы найдете в руководстве *10 Minutes to pandas*³.

Визуализация вовлеченности аудитории с помощью `matplotlib`

Теперь у нас есть инструменты, необходимые для измерения вовлеченности поклонников страницы в Facebook, но нам также нужна возможность легко сравнивать разные страницы (например, вместо музыкантов вы можете посмотреть, насколько хорошо страница вашей компании привлекает своих последователей, и сравнить ее показатели с показателями страниц ваших конкурентов).

Нам нужен способ, позволяющий легко агрегировать данные из нескольких источников в одну таблицу и манипулировать ею и, возможно, генерировать некоторые диаграммы из этих данных. В этом нам поможет библиотека `pandas` для Python. Она предлагает набор мощных структур данных и инструментов анализа, которые сделают вашу жизнь специалиста по данным или аналитика гораздо легче.

В следующем примере мы объединим данные со страниц тех же трех музыкантов, которые мы использовали в этой части главы, и сохраним их в `DataFrame` — табличной структуре данных, предлагаемой библиотекой `pandas`.

Чтобы установить библиотеку `pandas`, достаточно выполнить команду `pip install -U pandas` в командной строке. Флаг `-U` гарантирует установку самой свежей версии библиотеки.

Начнем, как показано в примере 2.8, с определения столбцов в пустой структуре `DataFrame`, где будут храниться наши данные.

Пример 2.8. Определение пустой структуры `DataFrame`

```
import pandas as pd # pip install pandas

# Создать экземпляр DataFrame для хранения информации,
# полученной из лент сообщений артистов
```

¹ <http://bit.ly/2C2k4gt>.

² <http://bit.ly/2BRE3vC>.

³ <http://bit.ly/2Dyd20w>.

```
columns = ['Name',
           'Total Fans',
           'Post Number',
           'Post Date',
           'Headline',
           'Likes',
           'Shares',
           'Comments',
           'Rel. Likes',
           'Rel. Shares',
           'Rel. Comments']
musicians = pd.DataFrame(columns=columns)
```

Здесь мы определили интересующие нас столбцы. Выполнив обход сообщений в лентах музыкантов, мы получим количество лайков, комментариев и ссылок, а также относительные показатели (то есть долю от общего числа поклонников, отреагировавших на сообщение). Как это сделать, показано в примере 2.9.

Пример 2.9. Сохранение данных в структуре DataFrame

```
# Собрать в структуру DataFrame последние 10 сообщений и оценки вовлеченности
# аудитории для каждого артиста
for page_id in [taylor_swift_id, drake_id, beyonce_id]:
    name = g.get_object(id=page_id)['name']
    fans = get_total_fans(page_id)
    feed = retrieve_page_feed(page_id, 10)
    for i, post in enumerate(feed):
        likes, shares, comments = measure_response(post['id'])
        likes_pct, shares_pct, comments_pct = measure_engagement(post['id'], fans)
        musicians = musicians.append({'Name': name,
                                      'Total Fans': fans,
                                      'Post Number': i+1,
                                      'Post Date': post['created_time'],
                                      'Headline': get_post_message(post),
                                      'Likes': likes,
                                      'Shares': shares,
                                      'Comments': comments,
                                      'Rel. Likes': likes_pct,
                                      'Rel. Shares': shares_pct,
                                      'Rel. Comments': comments_pct,
                                      }, ignore_index=True)

# Исправить типы данных некоторых столбцов
for col in ['Post Number', 'Total Fans', 'Likes', 'Shares', 'Comments']:
    musicians[col] = musicians[col].astype(int)
```

Мы перебираем идентификаторы страниц исполнителей, для каждой извлекаем имя, количество поклонников и последние 10 постов из ленты. Затем

внутренний цикл `for` перебирает 10 постов в каждой ленте и получает такие сведения, как общее количество лайков, комментариев и ссылок, а также вычисляет, какой процент от общего количества поклонников представляют эти числа. Вся эта информация записывается в `DataFrame` в виде записи. Передавая именованный аргумент `ignore_index` со значением `True`, мы указываем, что добавляемая запись не имеет предопределенного индекса, поэтому `pandas` будет индексировать записи порядковыми номерами по мере добавления.

Самый последний цикл в примере изменяет тип данных (`dtype`) некоторых столбцов, чтобы впоследствии `pandas` знала, что они хранят целые числа, а не числа с плавающей точкой или какие-то другие значения.

Выполнение этого кода может занять некоторое время, потому что данные извлекаются из Facebook через API, но в конечном итоге у нас появится прекрасная таблица, пригодная для дальнейших манипуляций. Структура `DataFrame` имеет удобный метод `head()`, с помощью которого можно просмотреть первые пять записей в таблице (рис. 2.6). Мы настоятельно рекомендуем проводить любые исследования в блокноте для Jupyter Notebook¹, таком как в репозитории GitHub для этой книги².

	Name	Total Fans	Post Number	Post Date	Headline	Likes	Shares	Comments	Rel. Likes	Rel. Shares	Rel. Comments
0	Taylor Swift	73862332	1	2017-12-19T17:07:33+0000	Check out a key moment in Taylor writing "This..."	33134	1994	1373	0.044859	0.002700	0.001859
1	Taylor Swift	73862332	2	2017-12-17T16:42:38+0000		6282	19	353	0.011213	0.000026	0.000478
2	Taylor Swift	73862332	3	2017-12-17T03:51:04+0000		11083	8	383	0.015004	0.000011	0.000519
3	Taylor Swift	73862332	4	2017-12-16T20:19:52+0000	The Swift Life is available for free worldwide...	36237	925	1012	0.053122	0.001252	0.001370
4	Taylor Swift	73862332	5	2017-12-15T13:18:45+0000	#TheSwiftLife App is available NOW for free in...	60721	1895	2105	0.082210	0.002566	0.002850

Рис. 2.6. Первые пять записей из `musicians` — структуры данных `DataFrame`

Структура `DataFrames` имеет несколько функций для построения графиков, которые упрощают визуализацию данных. За кулисами эти функции используют библиотеку `matplotlib`, поэтому обязательно установите ее.

Пример 2.10 демонстрирует одну из замечательных особенностей `pandas` — поддержку доступа к данным по индексу. Мы берем переменную `musicians` со структурой данных `DataFrame` и выбираем записи, в которых столбец `Name` хранит

¹ <http://bit.ly/2omIqdG>.
² <http://bit.ly/Mining-the-Social-Web-3E>.

имя Drake. То есть все дальнейшие операции выполняются только с данными, которые имеют отношение к Дрейку (Drake), и не затрагивают записи, соответствующие Тейлор Свифт (Taylor Swift) и Бейонсе (Beyoncé).

Пример 2.10. Вывод гистограммы на основе данных в DataFrame

```
import matplotlib # pip install matplotlib

musicians[musicians['Name'] == 'Drake'].plot(x='Post Number', y='Likes',
                                             kind='bar')
musicians[musicians['Name'] == 'Drake'].plot(x='Post Number', y='Shares',
                                             kind='bar')
musicians[musicians['Name'] == 'Drake'].plot(x='Post Number',
                                             y='Comments', kind='bar')
```

Этот код создаст гистограмму (рис. 2.7), отображающую количество лайков, которые получили последние 10 постов на странице Facebook Дрейка.

На рис. 2.7 сразу видно, что пост 8 был воспринят очень хорошо и может представлять для нас интерес. Также на рис. 2.8 видно, какая доля от общего числа поклонников среагировала на посты.

Пост 8 привлек 0,4% от общей аудитории, что, по всей видимости, относительно большое количество.

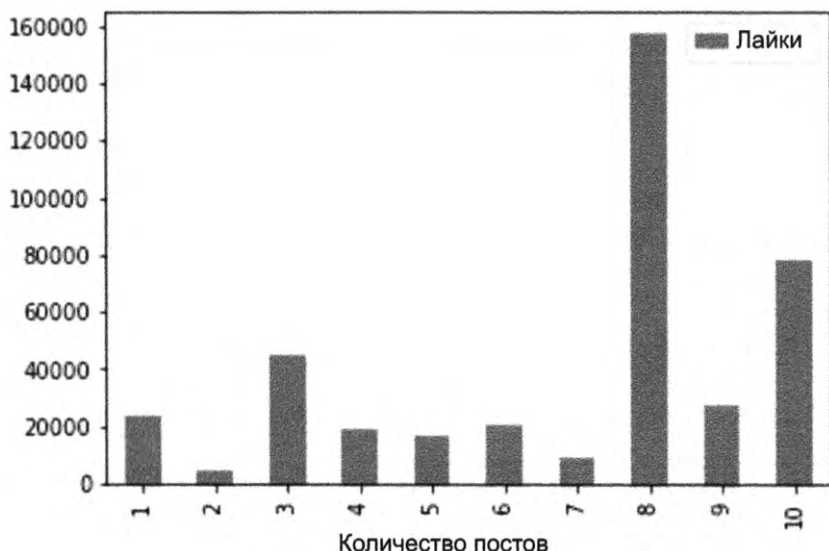


Рис. 2.7. Гистограмма с числом лайков для 10 последних постов

Теперь сравним трех исполнителей друг с другом. В настоящее время индекс в `DataFrame` — это самый обычный порядковый номер записи, но его можно заменить на что-то более значимое, что упростит манипулирование данными — мы немного изменим нашу структуру `DataFrame`, настроив *множественное индексирование*.

Множественный индекс — это индекс с иерархической организацией. В нашем случае верхним уровнем будет служить имя: Тейлор Свифт, Дрейк или Бейонсе. Уровнем ниже будет находиться номер столбца (от 1 до 10). Имя артиста и номера записи вместе однозначно определяют запись в нашей структуре `DataFrame`.

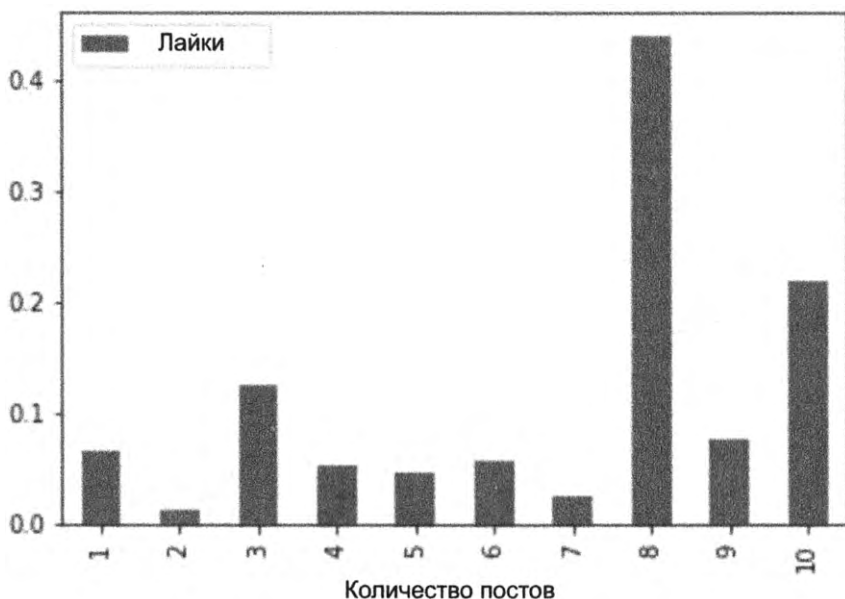


Рис. 2.8. Гистограмма с числом лайков для 10 последних постов, деленных на общее число фолловеров (поклонников)

Настройка множественного индексирования показана в примере 2.11.

Пример 2.11. Настройка множественного индексирования в `DataFrame`

```
# Преобразовать индекс в множественный индекс
musicians = musicians.set_index(['Name', 'Post Number'])
```

После настройки множественного индексирования появляется возможность выполнять мощные операции агрегирования данных с помощью метода `unstack`, как

демонстрирует пример 2.12. На рис. 2.9 показаны исходные записи в DataFrame после вызова метода `unstack`.

Пример 2.12. Использование метода `unstack` для поворота таблицы в DataFrame

```
# Метод unstack поворачивает таблицу в DataFrame
# и позволяет сгруппировать данные по столбцам, представляющим артистов
musicians.unstack(level=0)['Likes']
```

Name	Beyoncé	Drake	Taylor Swift
Post Number			
1	8328	23938	33134
2	18545	4474	8282
3	21589	44887	11083
4	10717	19003	39237
5	25497	17109	60721
6	17744	20328	41359
7	8934	9178	54012
8	10605	157515	29189
9	72254	27186	38439
10	10889	78674	33159

Рис. 2.9. Результат работы примера 2.12

Конечно, информация в таком виде лучше воспринимается визуально, поэтому используем встроенные операции для создания еще одной гистограммы, как показано в примере 2.13.

Пример 2.13. Создание гистограммы с общим числом лайков по артистам для последних 10 сообщений

```
# Вывести сравнительную гистограмму с реакциями на последние
# 10 сообщений артистов в Facebook
plot = musicians.unstack(level=0)['Likes'].plot(kind='bar', subplots=False,
                                                figsize=(10,5), width=0.8)
plot.set_xlabel('10 Latest Posts')
plot.set_ylabel('Number of Likes Received')
```

Получившаяся гистограмма показана на рис. 2.10.

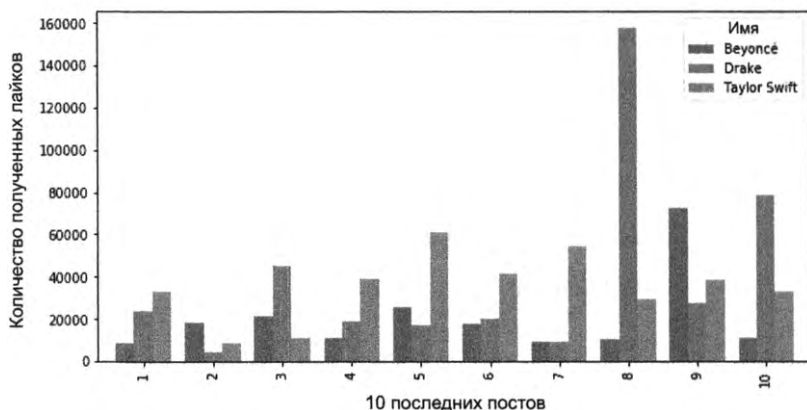


Рис. 2.10. Общее число лайков по исполнителям для последних 10 постов

Это очень удобный способ сравнения разных наборов данных друг с другом. Далее, поскольку каждый артист имеет свое число поклонников, давайте нормализуем количество лайков, приведя его к числу фолловеров, как показано в примере 2.14.

Пример 2.14. Создание гистограммы с относительным числом лайков по исполнителям для последних 10 постов

```
# Вывести гистограмму вовлеченности сообществ поклонников артистов
# для последних 10 сообщений
plot = musicians.unstack(level=0)['Rel. Likes'].plot(kind='bar', subplots=False,
                                                    figsize=(10,5), width=0.8)
plot.set_xlabel('10 Latest Posts')
plot.set_ylabel('Likes / Total Fans (%)')
```

Получившаяся гистограмма показана на рис. 2.11.

Обратите внимание: несмотря на то что у Дрейка (Drake) значительно меньше фолловеров в Facebook, чем у Бейонсе (Beyoncé) или Тейлор Свифт (Taylor Swift), многим его постам на странице Facebook удается вызвать реакцию у большей доли поклонников. Это может означать, что фанаты Дрейка более лояльны и активны в Facebook. Это также может означать, что Дрейк (или его представитель в социальных сетях) публикует в Facebook контент, очень привлекательный для фанов.

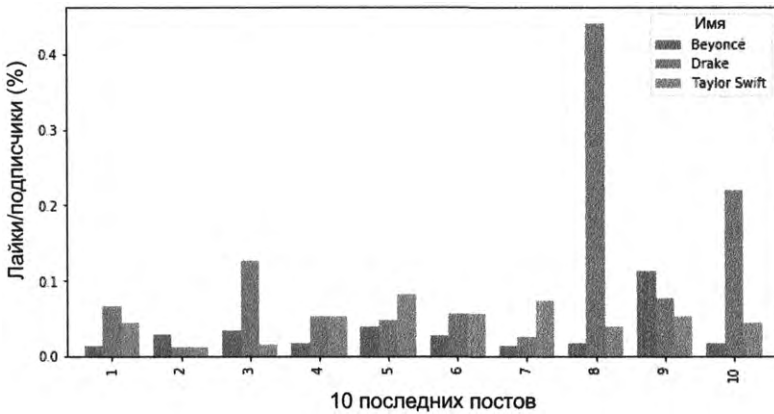


Рис. 2.11. Относительное число лайков по артистам для последних 10 постов

При более глубоком анализе можно было бы учесть содержимое каждого поста, тип контента (текст, изображение или видео), язык, используемый в комментариях, и какие другие реакции (кроме лайков) выражаются.

Вычисление средней вовлеченности

Еще одной полезной особенностью поддержки множественного индексирования в `DataFrame` является возможность вычисления статистики по индексу. Мы уже видели, как получить количество реакций на последние 10 постов для каждого из трех музыкантов. Точно так же можно вычислить среднее относительное количество лайков, комментариев или ссылок для каждого артиста, как показано в примере 2.15.

Пример 2.15. Вычисление средней вовлеченности для последних 10 постов

```
print('Average Likes / Total Fans')
print(musicians.unstack(level=0)['Rel. Likes'].mean())

print('\nAverage Shares / Total Fans')
print(musicians.unstack(level=0)['Rel. Shares'].mean())

print('\nAverage Comments / Total Fans')
print(musicians.unstack(level=0)['Rel. Comments'].mean())
```

Здесь снова используется метод `unstack` для агрегирования, чтобы сгруппировать каждый из исходных столбцов («Likes», «Shares», «Rel. Comments» и т. д.)

по именам артистов. Единственными записями в `DataFrame` теперь являются отдельные посты, отсортированные по порядковым номерам.

Пример 2.15 демонстрирует, как можно выбрать один из столбцов, например «`Rel. Likes`» (относительное число лайков), и найти среднее по этому столбцу.

Вот как выглядит вывод примера 2.15:

```
Average Likes / Total Fans
```

```
Name
```

```
Beyoncé          0.032084
```

```
Drake            0.112352
```

```
Taylor Swift     0.047198
```

```
dtype: float64
```

```
Average Shares / Total Fans
```

```
Name
```

```
Beyoncé          0.000945
```

```
Drake            0.017613
```

```
Taylor Swift     0.001962
```

```
dtype: float64
```

```
Average Comments / Total Fans
```

```
Name
```

```
Beyoncé          0.001024
```

```
Drake            0.016322
```

```
Taylor Swift     0.002238
```

```
dtype: float64
```

2.4. Заключительные замечания

Целью этой главы было познакомить вас с Graph API, показать, как протокол Open Graph может создавать связи между произвольными веб-страницами и социальным графом Facebook и как программно обращаться к социальному графу для анализа страниц Facebook и своей собственной социальной сети. Если вы опробовали примеры в этой главе, у вас не должно возникнуть никаких проблем с поиском ответов на вопросы, которые могут оказаться полезными. Но имейте в виду, что, занимаясь исследованием такого огромного и интересного набора данных, как социальный граф Facebook, вам нужно создать хорошую отправную точку. Изучение ответов на начальный запрос, скорее всего, натолкнет вас на естественный курс исследования,

в ходе которого вы последовательно будете улучшать свое понимание данных и приближаться к искомым ответам.

Возможности для добычи данных на Facebook огромны, но уважайте частную жизнь других и всегда соблюдайте условия предоставления услуг Facebook¹. В отличие от данных из Twitter и некоторых других источников, более открытых по своей природе, данные в Facebook могут иметь конфиденциальный характер, особенно если вы анализируете свою собственную социальную сеть. Надеюсь, в этой главе нам удалось показать, что есть много интересных возможностей для анализа социальных данных и в Facebook скрыто много ценной информации.



Исходный код примеров для этой и всех других глав доступен на GitHub² в удобном формате Jupyter Notebook, который вы можете опробовать, не покидая веб-браузера.

2.5. Упражнения

- Попробуйте извлечь данные из фан-страницы в Facebook кого-либо интересного для вас и попробуйте проанализировать язык, на котором написаны комментарии. Какие наиболее типичные темы обсуждаются на странице? Можете ли вы определить, когда поклонники довольны или расстроены чем-то?
- Выберите две разные фан-страницы, похожие по своей природе, и сравните их. Например, какие сходства и различия можно выделить между поклонниками сетей ресторанов Chipotle Mexican Grill и Taco Bell? Сможете ли вы найти что-нибудь необычное?
- Выберите знаменитость или компанию, очень активную в социальных сетях. Загрузите побольше публичных сообщений с ее страницы и оцените, насколько активно поклонники взаимодействуют с контентом. Есть ли какие-то закономерности? Какие посты вызвали наибольший интерес, если судить по количеству лайков, комментариев и ссылок?

¹ <http://on.fb.me/1a1lMXM>.

² <http://bit.ly/Mining-the-Social-Web-3E>.

Есть ли у топовых постов что-то общее? А у постов с низким уровнем вовлеченности?

- Количество объектов Facebook, доступных для Graph API, огромно. Можете ли вы исследовать объекты, такие как фотографии или сведения о посещениях, чтобы узнать больше о ком-либо в вашей сети? Например, кто публикует больше всего фотографий и как они характеризуются в комментариях? Где ваши друзья регистрируются чаще всего?
- Используйте гистограммы (представленные в разделе «Визуализация частот с помощью гистограмм» в главе 1) для дальнейшего анализа данных на странице Facebook. Создайте гистограмму распределения постов по времени суток. Они появляются в любое время дня и ночи? Или есть какое-то предпочтительное время суток для отправки?
- Дополнительно попробуйте оценить количество лайков, которые получают посты в зависимости от времени суток их публикации. Маркетологи в социальных сетях стремятся добиться максимального эффекта от сообщений и тщательно выбирают время их публикации, но учитывая работу алгоритма формирования лент новостей в Facebook, трудно сказать, когда именно ваша аудитория увидит пост.

2.6. Онлайн-ресурсы

Ниже приводится список ссылок, упоминавшихся в этой главе, которые могут оказаться полезными для вас:

- Facebook для разработчиков¹.
- Документация с описанием особенностей постраничного получения информации².
- Политика конфиденциальности платформы Facebook³.
- Graph API Explorer⁴.
- Обзор Graph API⁵.

¹ <http://bit.ly/1a1lm3Q>.

² <http://bit.ly/1a1ltMP>.

³ <http://bit.ly/1a1lb3C>.

⁴ <http://bit.ly/2jd5Xdq>.

⁵ <http://bit.ly/1a1lobU>.

- Справочник Graph API¹.
- HTML-элементы meta².
- OAuth³.
- Протокол Open Graph⁴
- Библиотека requests для Python⁵.
- RDFa⁶.

¹ <http://bit.ly/1a1vEr>.

² <http://bit.ly/1a1BMa>.

³ <http://bit.ly/1a1kZWN>.

⁴ <http://bit.ly/1a1lu3m>.

⁵ <http://bit.ly/1a1lrEt>.

⁶ <http://bit.ly/1a1ujR>.

3 Instagram: компьютерное зрение, нейронные сети, распознавание объектов и лиц

В предыдущих главах основное внимание уделялось анализу текстовых данных, полученных из социальных сетей, структуре самих сетей и оценке вовлеченности людей во взаимодействие с контентом. Instagram как приложение социальной сети в первую очередь служит для обмена изображениями и видео. Оно появилось в 2010 году и быстро завоевало популярность. Приложение позволяет легко редактировать фотографии и применять различные фильтры. А так как оно изначально предназначалось для использования на смартфонах, приложение дало простой способ делиться фотографиями с остальным миром.

Менее чем через два года после появления компания Facebook приобрела Instagram, и в июне 2018 года приложение достигло ошеломляющего уровня в 1 миллиард пользователей, что сделало его одной из самых популярных социальных сетей в мире.

По мере расширения социальных сетей технологические компании продолжают искать новые способы извлечения ценности из данных, выгружаемых на их платформы. Например, такие компании, как Facebook и Google, активно нанимают людей с опытом в области машинного обучения, то есть с опытом обучения компьютеров распознаванию самых разных закономерностей в данных.

Машинному обучению можно найти множество применений, например: определение, какой контент вам понравится, какие объявления вызовут у вас интерес, какие слова лучше использовать для исправления ошибок, которые вы допускаете при наборе текста неуклюжими пальцами, и т. д. Как и многое другое, машинное обучение может использоваться в неблагоприятных целях, поэтому важно быть в курсе его применений и выступать за соблюдение норм и правил при использовании данных.

Одним из захватывающих технологических прорывов последних лет является разработка значительно улучшенных алгоритмов компьютерного зрения. Эти новейшие алгоритмы используют глубокие нейронные сети, обученные распознаванию объектов на изображениях, что имеет огромное значение, например, для создания беспилотных транспортных средств. Искусственные нейронные сети — алгоритмы машинного обучения, которые требуют большого количества эталонных образцов и могут быть обучены распознаванию любых закономерностей в данных разных видов, включая изображения.

Поскольку Instagram является социальной сетью с акцентом на обмен фотографиями, одним из лучших способов анализа таких данных является применение нейронных сетей. Первый вопрос, на который мы пытаемся ответить: что это за фотография? Есть ли на ней горы? Озера? Люди? Машины? Животные? Алгоритмы этого типа также можно использовать для фильтрации незаконного контента или материалов для взрослых, размещенных в интернете, и позволить модераторам справиться с потоком данных.

В этой главе вы узнаете, как работают нейронные сети. Мы построим свою простую нейронную сеть для распознавания рукописных цифр. На Python эта задача решается довольно просто благодаря нескольким мощным библиотекам машинного обучения. Затем мы используем Google Vision API для распознавания объектов и лиц на фотографиях в лентах Instagram. Поскольку мы создадим всего лишь небольшое тестовое приложение для доступа к Instagram API, вы сможете использовать только фотографии из своей ленты, поэтому для опробования примеров вам понадобится аккаунт в Instagram и, по крайней мере, несколько фотографий, опубликованных на этой платформе.



Загрузите последний исправленный исходный код примеров для этой (и любой другой) главы, доступный по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Воспользуйтесь также преимуществами виртуальной машины, описанными в приложении А, чтобы получить максимальное удовольствие от опробования примеров кода.

3.1. Обзор

В этой главе мы познакомимся с машинным обучением, которое в последнее время привлекает много внимания из-за его применения в искусственном интеллекте. Возникли тысячи новых проектов, применяющих машинное обучение для решения различных задач. В этой главе мы посмотрим, как

эту технологию применить к изображениям, извлеченным из социальной медиаплатформы, ориентированной на обмен изображениями. В частности, вы познакомитесь с:

- программным интерфейсом Instagram и узнаете, как к нему обращаться;
- структурой данных, возвращаемых программным интерфейсом Instagram;
- основными идеями, лежащими в основе нейронных сетей;
- приемами использования нейронных сетей для «просмотра» изображений и распознавания объектов на них;
- возможностью применения мощных, предварительно обученных нейронных сетей для распознавания объектов и лиц на изображениях в вашей ленте сообщений Instagram.

3.2. Instagram API

Чтобы получить доступ к Instagram API, необходимо создать приложение и зарегистрировать его. Это легко сделать, воспользовавшись платформой разработчиков Instagram¹. Просто зарегистрируйте нового клиента, указав имя и описание (например, «мое тестовое приложение»), а также URL для переадресации на веб-сайт (например, www.google.com). Зачем нужен URL переадресации, вы узнаете чуть позже.

Новый клиент всегда действует в *безопасном окружении*² (*песочнице*), ограничивающем его возможности. Приложения в песочнице ограничены 10 пользователями и 20 последними медиафайлами, опубликованными каждым из этих пользователей. Кроме того, API накладывает строгие ограничения на частоту обращений к нему. Эти ограничения позволяют протестировать приложение перед отправкой на проверку. Сотрудники Instagram могут проверить ваше приложение и, если оно соответствует разрешенным случаям использования, — одобрить его и снять эти ограничения. Поскольку все свое внимание мы сосредоточим на изучении API, все наши примеры предполагают работу в безопасном режиме. Цель этой главы — познакомить вас с основными инструментами, чтобы вы могли продолжить обучение самостоятельно и создавать более сложные приложения.

¹ <http://bit.ly/1rbjGmz>.

² <http://bit.ly/2Ia88Nr>.



На момент написания этой главы, в начале 2018 года, многие платформы социальных сетей находились под все более расширяющимся контролем общественности над тем, как они обрабатывают данные и сколько сторонних приложений могут получить доступ к данным пользователей. Несмотря на то что большая часть данных, размещенных в Instagram, является общедоступной, создатели Instagram продолжают ужесточать правила доступа к программному интерфейсу, и когда вы будете читать эти строки, некоторые функции могут оказаться устаревшими. Поэтому за самой свежей информацией всегда обращайтесь к документации для разработчика¹.

3.2.1. Выполнение запросов к Instagram API

На рис. 3.1 показана страница, которая откроется после регистрации нового клиента, действующего в безопасном окружении. Теперь вы практически готовы начать выполнять запросы к API. На странице управления клиентами² щелкните на кнопке **Manage** (Управлять), чтобы перейти к управлению вновь созданным приложением, затем скопируйте идентификатор и секрет клиента в объявления переменных в примере 3.1. Также скопируйте URL-адрес веб-сайта, объявленного вами при регистрации клиента, и вставьте его в объявление переменной `REDIRECT_URI` в примере 3.1.

Пример 3.1. Аутентификация в Instagram API

Подставьте свои идентификатор клиента, секрет клиента и URL для переадресации

```
CLIENT_ID = ''
CLIENT_SECRET = ''

REDIRECT_URI = ''

base_url = 'https://api.instagram.com/oauth/authorize/'

url='{}?client_id={}&redirect_uri={}&response_type=code&scope=public_content'\
    .format(base_url, CLIENT_ID, REDIRECT_URI)

print('Click the following URL, \
which will take you to the REDIRECT_URI \
you set in creating the APP.')
print('You may need to log into Instagram.')
print()
print(url)
```

¹ <http://bit.ly/2Ibb4JL>.

² <http://bit.ly/2IayH4Z>.

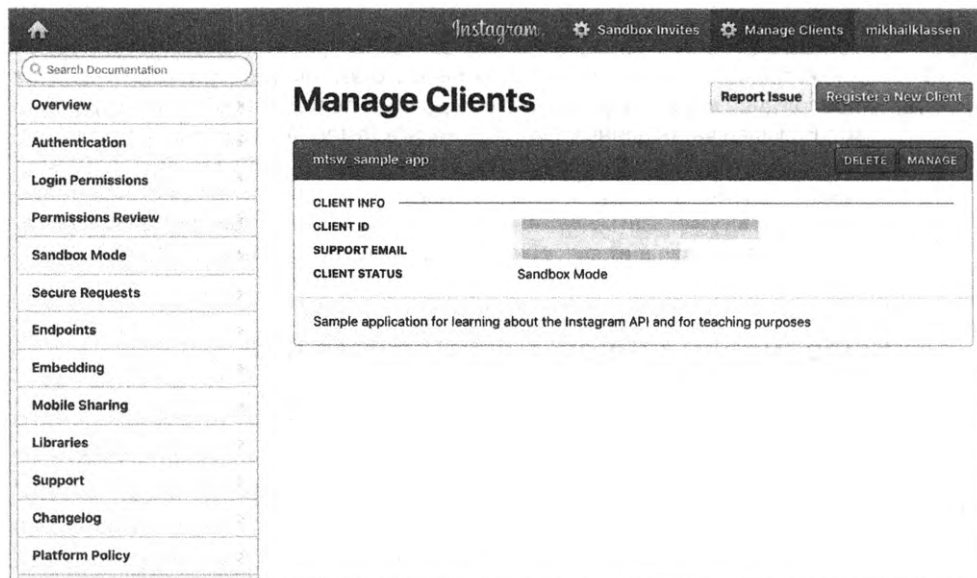


Рис. 3.1. Страница управления клиентом на платформе разработчиков Instagram с нашим приложением в песочнице

В процессе выполнения этот код выведет примерно следующее¹ (с уникальными значениями в параметрах `client_id` и `redirect_uri` в URL):

Click the following URL, which will take you to the REDIRECT_URI you set in creating the APP.

You may need to log into Instagram.

https://api.instagram.com/oauth/authorize/?client_id=...&redirect_uri=...&response_type=code&scope=public_content

Вывод содержит адрес URL, который можно скопировать и вставить в адресную строку веб-браузера. После перехода по этой ссылке вам будет предложено выполнить вход в свой аккаунт в Instagram, и затем произойдет переадресация на URL веб-сайта, который вы указали при регистрации своего клиента. В конец URL будет добавлен специальный параметр `?code= ....` Скопируйте и вставьте этот код в пример 3.2, чтобы завершить процесс аутентификации и получить доступ к Instagram API.

¹ Перевод: «Щелкните на следующей ссылке, которая переадресует вас по REDIRECT_URI, указанному в приложении. При этом вам может потребоваться выполнить вход в Instagram». — *Примеч. пер.*

Пример 3.2. Получение токена доступа

```
import requests # pip install requests

CODE = ''

payload = dict(client_id=CLIENT_ID,
               client_secret=CLIENT_SECRET,
               grant_type='authorization_code',
               redirect_uri=REDIRECT_URI,
               code=CODE)

response = requests.post(
    'https://api.instagram.com/oauth/access_token',
    data = payload)

ACCESS_TOKEN = response.json()['access_token']
```

Код в примере 3.2 сохранит специальный токен доступа в переменную `ACCESS_TOKEN`, после чего его можно использовать для выполнения вызовов к API.

Наконец, протестируйте приложение, для чего попробуйте получить метаданные из своего профиля в Instagram. Для этого выполните код в примере 3.3, который должен вернуть ответ в формате JSON, включающий имя пользователя, местоположение изображения профиля и биографию, а также информацию о количестве ваших сообщений, количестве пользователей, за которыми вы следуете, и количестве пользователей, которые следуют за вами.

Пример 3.3. Подтверждение доступа к платформе с использованием токена доступа, полученного в примере 3.2

```
url = 'https://api.instagram.com/v1/users/self/?access_token='
response = requests.get(url+ACCESS_TOKEN)
print(response.text)
```

Вот как примерно выглядит этот ответ:

```
{"data": {"id": "...", "username": "mikhailklassen", "profile_picture":
"https://scontent.cdninstagram.com/vp/bf2fed5bbce922f586e55db2944fdc9c/5B908514
/t51.2885-19/s150x150/22071355_923830121108291_7212344241492590592_n.jpg",
"full_name": "Mikhail Klassen", "bio": "Ex-astrophysicist, entrepreneur,
traveler,
wine \u0026 spirits geek.\nPhotography: #travel #architecture #art #urban
#outdoors #wine #spirits", "website": "http://www.mikhailklassen.com/",
"is_business": false, "counts": {"media": 162, "follows": 450, "followed_by":
237}}, "meta": {"code": 200}}
```

3.2.2. Извлечение своей ленты постов из Instagram

В отличие от многих других глав этой книги, здесь мы не будем использовать специальную библиотеку Python для доступа к ленте Instagram. На момент написания этих строк компания Instagram проводила реорганизацию своего API, и ее новый Graph API¹, вероятно, многое позаимствует из стека технологий, используемого материнской компанией Facebook в своем Graph API (см. главу 2). По этой причине одна из официальных Python-библиотек для Instagram была законсервирована.

Вместо этого мы используем библиотеку `requests`² для Python. Она входит в состав стандартной библиотеки, и поскольку Instagram поддерживает RESTful API³, с ее помощью мы сможем выполнять запросы к API по протоколу HTTP. Еще одно преимущество заключается в том, что код, который мы напишем, будет очень прозрачным, он не будет скрывать деталей происходящего за абстрактной оберткой. И, как вы увидите, для доступа к Instagram API требуется не так много строк кода.

Пример 3.4 извлекает вашу собственную ленту с помощью всего одной строки кода на Python, используя метод `requests.get()`, посылая запрос конечной точке API, представляющей ленту, и выполняя аутентификацию с помощью токена доступа, полученного ранее. Остальная часть примера содержит определение функции для отображения ленты с изображениями. Этот пример должен запускаться в Jupyter Notebook и использует виджеты IPython для отображения встроенных изображений.

Пример 3.4. Отображение изображений и подписей к ним из вашей ленты в Instagram

```
from IPython.display import display, Image
url = 'https://api.instagram.com/v1/users/self/media/recent/?access_token='
response = requests.get(url+ACCESS_TOKEN)
recent_posts = response.json()

def display_image_feed(feed, include_captions=True):
    for post in feed['data']:
        display(Image(url=post['images']['low_resolution']['url']))
        print(post['images']['standard_resolution']['url'])
```

¹ <http://bit.ly/2jTGHce>.

² <http://bit.ly/1a1lrEt>.

³ <http://bit.ly/2rC8oJW>.

```
if include_captions: print(post['caption']['text'])
print()
```

```
display_image_feed(recent_posts, include_captions=True)
```

Ответ, возвращаемый Instagram API в формате JSON, содержит не сами изображения, а ссылки на них. Для обеспечения высокой производительности Facebook и Instagram размещают контент в сети доставки контента¹ — в дата-центрах, географически разбросанных по всему миру, чтобы обеспечить быструю доставку.

Код из примера 3.4 выведет изображения с их адресами URL и подписями, как показано на рис. 3.2.



https://scontent.cdninstagram.com/vp/d865700e40b05f30ed74e5f38e574a81/5B94F2E2/t51.2885-15/e640x640/sh0.08/e35/30855391_1542497469196486_3914453959741276160_n.jpg
The Boston Public Library on what felt like the first real day of Spring. #bpl #boston #copleysquare #library



https://scontent.cdninstagram.com/vp/3cd8f6420345043fcc9b2d6f767afd7d/5B7C3E0C/t51.2885-15/e640x640/sh0.08/e35/30078582_1348310691937848_7383458251121098752_n.jpg
On top of Montreal: the view from the terrace at the Place Ville-Marie restaurant Les Enfants Terribles. #MTL #PVM #Montreal #MountRoyal #travel

Рис. 3.2. Пример вывода, произведенного примером 3.4, с двумя последними постами из ленты в Instagram, включающими адреса URL изображений и их подписи

¹ <http://bit.ly/2Gb0DzH>.

3.2.3. Извлечение медиафайлов по хештегу

Для отправки запросов к конечной точке API, имеющей форму URL (например, https://api.instagram.com/v1/users/self/media/recent/?access_token=...), в предыдущих примерах использовалась библиотека `requests`.

Структура этого URL сообщает Instagram API, какие данные необходимо вернуть, а токен доступа обеспечивает требуемую аутентификацию. Полное описание всех конечных точек¹ и типов информации, которые они возвращают, вы найдете в документации для разработчиков.

Код в примере 3.4 извлечет 20 последних сообщений из вашей личной ленты в Instagram (максимум, разрешенный для приложений, действующих в песочнице). Для анализа данных было бы интересно организовать фильтрацию ленты по хештегам пользователя, которые широко используются в Instagram.

Хештеги начали использоваться в Twitter примерно в 2007 году, как средство группировки твитов, а в 2009-м они были преобразованы в гиперссылки, чтобы дать возможность искать посты с этими хештегами. Они также используются для выявления актуальных тем и оказались универсальным инструментом. Благодаря хештегам пользователи могут добавлять свои метаданные, вносить свой вклад в конкретные беседы, протекающие на платформе, и вовлекать в обсуждения более широкоую аудиторию.

Instagram заимствовала хештеги и внедрила их в свою платформу, благодаря чему у пользователей появилась возможность искать и находить посты с конкретными хештегами.

Для поиска по хештегу в Instagram используется другая конечная точка API — с URL, содержащим искомый хештег, как показано в примере 3.5.

Пример 3.5. Поиск медиафайлов по хештегу

```
hashtag = 'travel'
response = requests.get('https://api.instagram.com/v1/tags/'
                        +hashtag+'/media/recent?access_token='
                        +ACCESS_TOKEN)

display_image_feed(response.json(), include_captions=True)
```

Этот пример выполняет поиск по популярному хештегу `#travel` и использует функцию `display_image_feed` из примера 3.4. Вывод этого примера напоминает

¹ <http://bit.ly/2I9iAEF>.

вывод на рис. 3.2, но отображает только посты, содержащие хештег `#travel`. Напомним еще раз, что приложение в песочнице вернет только медиафайлы из вашей собственной ленты, и только из последних 20 сообщений.

3.3. Анатомия поста в Instagram

Как мы видели выше, Instagram API возвращает ответы в формате JSON. Этот формат позволяет API вернуть структурированные данные в виде удобочитаемой иерархической коллекции пар атрибут/значение. Полное описание структуры данных можно найти в документации для разработчиков Instagram¹, а здесь мы лишь кратко рассмотрим их.

Содержимое ответа API мы будем выводить на экран с помощью библиотеки `json`. Соответствующий код показан в примере 3.6.

Пример 3.6. Вывод ответа API с помощью библиотеки `json`

```
import json

uri = ('https://api.instagram.com/v1/users/self/media/recent/?access_token='
      response = requests.get(uri+ACCESS_TOKEN)

print(json.dumps(recent_posts, indent=1))
```

Ниже приводится пример вывода, полученного с помощью этого кода:

```
{
  "pagination": {},
  "data": [
    {
      "id": "1762766336475047742_1170752127",
      "user": {
        "id": "1170752127",
        "full_name": "Mikhail Klassen",
        "profile_picture": "https://...",
        "username": "mikhailklassen"
      },
      "images": {
        "thumbnail": {
          "width": 150,
          "height": 150,
          "url": "https://...jpg"
```

¹ <http://bit.ly/2I9iAEF>.

```
    },
    "low_resolution": {
      "width": 320,
      "height": 320,
      "url": "https://...jpg"
    },
    "standard_resolution": {
      "width": 640,
      "height": 640,
      "url": "https://...jpg"
    }
  },
  "created_time": "1524358144",
  "caption": {
    "id": "17912334256150534",
    "text": "The Boston Public Library...#bpl #boston #copleysquare #library",
    "created_time": "1524358144",
    "from": {
      "id": "1170752127",
      "full_name": "Mikhail Klassen",
      "profile_picture": "https://...jpg",
      "username": "mikhailklassen"
    }
  },
  "user_has_liked": false,
  "likes": {
    "count": 15
  },
  "tags": [
    "bpl",
    "copleysquare",
    "library",
    "boston"
  ],
  "filter": "Reyes",
  "comments": {
    "count": 1
  },
  "type": "image",
  "link": "https://www.instagram.com/p/Bh2mqS7nKc-/",
  "location": {
    "latitude": 42.35,
    "longitude": -71.076,
    "name": "Copley Square",
    "id": 269985898
  },
  "attribution": null,
  "users_in_photo": []
},
```

```
{
  ...
}
],
"meta": {
  "code": 200
}
}
```

Вы могли бы подумать, что своей иерархической структурой этот ответ очень похож на словарь в языке Python, и это действительно так и есть.

Атрибутами верхнего уровня ответа являются `meta`, `data` и `pagination`. Атрибут `meta` содержит информацию о самом ответе. Если запрос обработан без ошибок, вы увидите в нем только один ключ `code` со значением `200` («ОК»). Но если что-то пошло не так, например, если вы отправили неверный токен доступа, вы получите исключение с соответствующим сообщением об ошибке.

Ключ `data` содержит фактический ответ. Его значение может иметь форму JSON, как показано в примере выше. Данные, получаемые в ответ на запрос из примера 3.6, содержат информацию о пользователе, разместившем медиафайл (включая имя пользователя, полное имя, идентификатор и изображение профиля), изображения (адреса URL) с разными разрешениями, а также сведения о самом сообщении (например, когда оно было создано, текст подписи, хештеги, применявшиеся фильтры, количество лайков и иногда — географическое местоположение).

В нашем случае приложение, действующее в песочнице, вернуло 20 изображений, и вся соответствующая информация содержится в атрибуте `data`. Приложение, одобренное в Instagram и действующее за пределами песочницы, наверняка вернуло бы более 20 результатов. Часто нежелательно, чтобы вся эта информация возвращалась в виде единого ответа, поэтому разработчики реализовали постраничный доступ к результатам.

Постраничный доступ¹ подразумевает деление данных на блоки разумного размера. Ответ API содержит атрибут `pagination`, значением которого является URL, ссылающийся на следующую «страницу» данных. Обращение к этому URL вернет следующий фрагмент ответа, который также будет содержать атрибут `pagination`, ссылающийся на следующий фрагмент, и так далее.

Извлекая URL изображений из атрибута `data` в ответе API, можно получить сами изображения и применить к ним методы анализа данных. Платформа

¹ <http://bit.ly/2Kio0K6>.

Instagram поощряет публичный обмен медиафайлами, так же как Twitter. В Instagram можно создавать закрытые аккаунты и открывать доступ к постам в них только явно перечисленным пользователям, однако подавляющее большинство аккаунтов являются публичными, а это означает (как вы уже знаете), что изображения тоже являются публичными.

3.4. Краткое введение в искусственные нейронные сети

Анализ изображений имеет долгую историю, но способность компьютеров рассматривать изображения¹ и распознавать предметы (например, собак или автомобили) является более поздним технологическим достижением.

Представьте, что вы должны описать робота, как выглядит собака. Вы могли бы описать ее как животное с четырьмя ногами, торчащими ушами, двумя глазами, большими зубами и т. д. Робот будет добросовестно применять эти правила и вернет тысячи ошибочных изображений других животных, также с четырьмя ногами, торчащими ушами и т. д. Кроме того, робот может не определить настоящих собак просто потому, что у них уши не торчащие, а висячие.

Если бы у робота была возможность узнать о собаках столько же, сколько знают люди, то есть если бы он мог увидеть много примеров собак, как мы с самого раннего детства, когда родители показывали нам на одну из них и говорили: «собака»!

В мозге младенца растет и развивается сеть биологических нейронов², образующих и разрывающих связи. Обратная связь с окружающей средой усиливает ключевые связи, в то время как другие атрофируются. Увидев достаточное количество собак, например, мы получаем довольно четкое представление о том, что такое собака, даже если нам трудно выразить эту идею в словах.

Искусственные нейронные сети³ были созданы по образу и подобию биологических нейронных сетей, существующих в нервной системе многих организмов. Нейронную сеть можно рассматривать как систему обработки информации с входами и выходами. Искусственная нейронная сеть обычно состоит из *слоев нейронов*, как показано на рис. 3.3. Каждый нейрон определяется *функцией*

¹ <http://bit.ly/2IygU79>.

² <http://bit.ly/2KWaOvR>.

³ <http://bit.ly/2IcVrkk>.

активации, которая имеет несколько входов, взвешиваемых некоторыми значениями, и отображает их в результат — обычно в значение от 0 до 1.

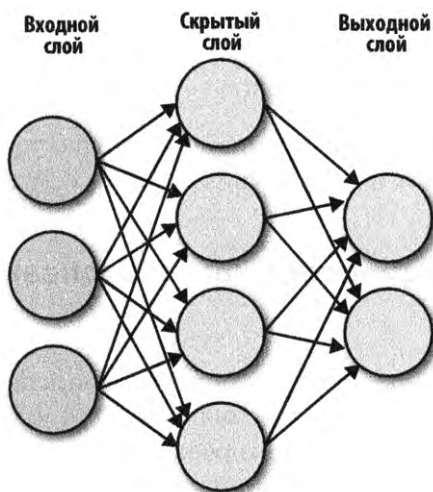


Рис. 3.3. Схематическое представление искусственной нейронной сети, имеющей входной слой с тремя нейронами, один скрытый слой с четырьмя нейронами и выходной слой с двумя нейронами (изображение представлено пользователем Glosser.ca на условиях лицензии CC BY-SA 3.0 и взято из Wikimedia Commons)

Выходы одного слоя связаны со входами следующего за ним. Нейронная сеть имеет *входной слой*, каждый нейрон которого соответствует одному интересующему признаку. Это могут быть, например, значения пикселей в изображении.

Выходной слой часто состоит из нейронов, представляющих классы. Например, можно создать сеть с двумя выходными нейронами, которая будет определять, присутствует ли на изображении кошка или нет. Это бинарный классификатор. Точно так же сеть может иметь много выходных нейронов, представляющих отдельные объекты, которые могут присутствовать на изображении.

В большинстве нейронных сетей между входным и выходным слоями имеется также один или несколько *скрытых слоев*. Их наличие делает внутреннюю работу сети менее прозрачной для нашего понимания, но они жизненно важны для обнаружения сложных нелинейных структур, которые могут присутствовать во входных шаблонах.

Чтобы добиться приемлемого уровня точности, нейронную сеть нужно «обучить». Обучение — это процесс, посредством которого сеть сравнивает входные

и спрогнозированные ею выходные данные с истинными. Разница между спрогнозированными и истинными данными представляет ошибку (также называемую *потерей*¹), которую сеть должна минимизировать. Используя алгоритм обратного распространения², сеть корректирует весовые коэффициенты на входах каждого нейрона. Выполнив последовательность итераций, сеть может уменьшить общую ошибку и улучшить точность прогнозирования.

В действительности существует огромное число тонкостей, не упомянутых здесь, но в интернете вы найдете массу хороших ресурсов, если захотите пойти дальше.

3.4.1. Обучение нейросети «рассматриванию» изображений

Впервые искусственные нейронные сети были применены почтовой службой США для распознавания почтовых индексов. Для ускорения доставки конверты с письмами требуется рассортировать, для чего необходимо прочитать почтовые индексы на них, часто написанные от руки. В США *почтовые индексы* представлены пяти- и девятизначным числами³. Первые пять цифр соответствуют географическому району, а дополнительные четыре — определенному маршруту доставки.

Для чтения индексов и сортировки почты в почтовой службе США используется технология оптического распознавания символов⁴ (optical character recognition, OCR). Есть несколько подходов к реализации компьютерного зрения, которые можно использовать для перевода изображений рукописных цифр в компьютерные символы. Так случилось, что искусственные нейронные сети особенно хорошо справляются с этой задачей.

Фактически, одной из первых задач, которую учатся решать студенты, изучающие курс машинного обучения и нейронных сетей, является задача распознавания рукописных цифр. Для оказания помощи в исследованиях нейронных сетей и сравнения различных алгоритмов компьютерного зрения был создан набор исследовательских данных, называемый базой данных MNIST⁵. На рис. 3.4 показан образец изображений в этой базе данных.

¹ <http://bit.ly/2KdyVoo>.

² <http://bit.ly/2jRgYRB>.

³ <http://bit.ly/2IuPpvt>.

⁴ <http://bit.ly/2IjnqLX>.

⁵ <http://bit.ly/2IaAxmC>.

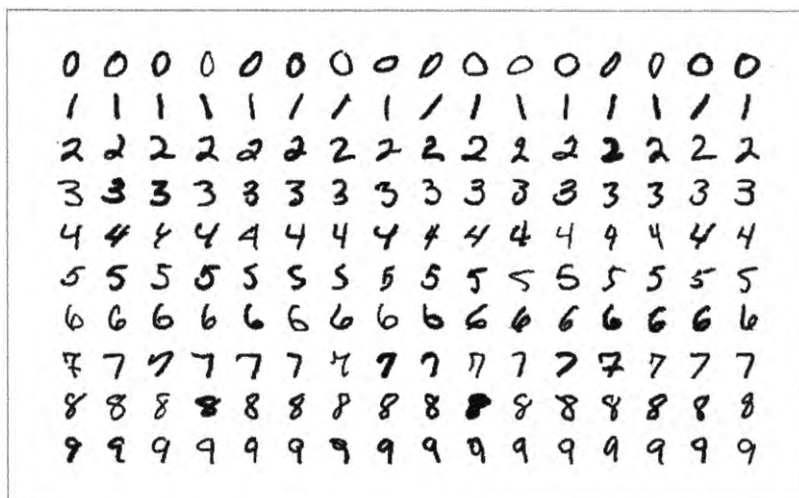


Рис. 3.4. Образец изображений из набора данных MNIST (изображение представлено Йозефом Степпаном (Josef Steppan) на условиях лицензии CC BY-SA 4.0 и взято из Wikimedia Commons)

База данных MNIST содержит 60 000 обучающих и 10 000 тестовых изображений. Существует также расширенная версия базы данных, называемая EMNIST¹; она содержит еще больше изображений и дополняет данные из MNIST изображениями рукописных букв.

Каждое изображение в базе данных MNIST состоит из 28×28 пикселей в оттенках серого. Значения пикселей, обычно нормализованные и приведенные к диапазону от 0 до 1, должны передаваться алгоритму, результатом которого является одна из цифр от 0 до 9.

Что для нейронной сети означает «рассмотреть» изображение? Если для каждого пиксела предусмотреть один входной нейрон, нам требуется 784 входных нейрона (28×28). Любое изображение может представлять только 1 из 10 возможных цифр, поэтому выходной слой должен иметь 10 нейронов, выходным значением каждого из которых является прогнозируемая вероятность, что изображение представляет соответствующую цифру. В качестве наилучшего прогноза нейронной сети возвращается цифра с наибольшей прогнозируемой вероятностью.

¹ <http://bit.ly/2rAZrS4>.

Между входным и выходными слоями можно добавить «скрытые» слои. Нейроны в этих слоях подобны любым другим, но на входе принимают результаты, полученные предыдущими слоями.

С этого момента задача быстро начинает обрастать техническими деталями, тогда как наша цель — начать применять нейронные сети для анализа изображений. Если эта тема вам интересна, могу порекомендовать книгу Орельена Жерона (Aurélien Géron) *Hands-On Machine Learning with Scikit-Learn and TensorFlow*¹ (O'Reilly) или главу 1 из онлайн-книги Майкла Нильсена (Michael Nielsen) *Neural Networks and Deep Learning*².

3.4.2. Распознавание рукописных цифр

Код в примере 3.7 создает многослойную нейронную сеть³ для классификации рукописных цифр, используя библиотеку `scikit-learn` для Python. Изображения рукописных цифр можно быстро загрузить с помощью вспомогательной функции. Эти изображения имеют более низкое разрешение, чем изображения в базе данных MNIST, и имеют размеры 8×8 пикселей. Всего в базе данных `scikit-learn` содержится 1797 изображений.



Установить библиотеку `scikit-learn` можно командой `pip install scikit-learn`, а необходимую ей библиотеку `scipy` (зависимость) — командой `pip install scipy`.

Пример 3.7. Использование классификатора многослойного перцептрона (Multilayer Perceptron, MLP) из библиотеки `scikit-learn` для распознавания рукописных цифр

```
# Установите scikit-learn и scipy (зависимость) с помощью команд:
# pip install scikit-learn
# pip install scipy
from sklearn import datasets, metrics
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split

digits = datasets.load_digits()
```

¹ <http://oriel.ly/2KV4X5S> (Жерон О. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. Концепции, инструменты и техники для создания интеллектуальных систем. М.: Вильямс, 2018. — *Примеч. пер.*)

² <http://bit.ly/2IjE1Pm>.

³ <http://bit.ly/2Ie80ME>.

```
# Масштабируйте исходные данные и разбейте на обучающую и контрольную выборки
X, y = digits.data / 255., digits.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
                                                    random_state=42)

mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=100, alpha=1e-4,
                    solver='adam', verbose=10, tol=1e-4, random_state=1,
                    learning_rate_init=.1)

mlp.fit(X_train, y_train)
print()
print("Training set score: {}".format(mlp.score(X_train, y_train)))
print("Test set score: {}".format(mlp.score(X_test, y_test)))
```

Далее приводится пример вывода этого кода:

```
Iteration 1, loss = 2.08212650
Iteration 2, loss = 1.03684958
Iteration 3, loss = 0.46502758
Iteration 4, loss = 0.29285682
Iteration 5, loss = 0.22862621
Iteration 6, loss = 0.18877491
Iteration 7, loss = 0.15163667
Iteration 8, loss = 0.13317189
Iteration 9, loss = 0.11696284
Iteration 10, loss = 0.09268670
Iteration 11, loss = 0.08840361
Iteration 12, loss = 0.08064708
Iteration 13, loss = 0.06800582
Iteration 14, loss = 0.06649765
Iteration 15, loss = 0.05651331
Iteration 16, loss = 0.05649585
Iteration 17, loss = 0.06339016
Iteration 18, loss = 0.06884457
Training loss did not improve more than tol=0.000100 for two consecutive epochs.
Stopping.

Training set score: 0.9806978470675576
Test set score: 0.9577777777777777
```

Обратите внимание, как в примере 3.7 выполняется деление набора данных на «обучающую» и «контрольную» выборки с помощью метода `train_test_split`. Эта функция случайным образом распределяет данные в две выборки, одну для обучения и одну для тестирования модели, руководствуясь заданным размером контрольной выборки. В нашем примере он равен `0.25`, то есть четверть данных отводится для тестирования.

В начале любой задачи машинного обучения очень важно отделить часть данных для оценки эффективности алгоритма. Это поможет избежать переобучения¹. Алгоритм не увидит данных из контрольной выборки во время обучения, благодаря этому вы сможете достаточно достоверно оценить точность алгоритма, запустив распознавание с контрольными данными.

Как можно заключить по результатам выполнения нашего кода, мы достигли почти 96% точности на контрольной выборке. Проще говоря, количество ошибок составляет около 4%. Не так уж плохо! Однако лучшим алгоритмам², использующим специальный тип глубокого обучения, который называют сверхточной нейронной сетью³, удастся достичь еще более высокой точности, когда количество ошибок на данных из MNIST не превышает 0,2%.

Обратите также внимание, что оценка точности на обучающей выборке выше оценки на контрольной выборке. Точность на обучающей выборке составила чуть более 98%. Для моделей машинного обучения характерна ситуация, когда на обучающей выборке они показывают лучшие результаты, чем на контрольной. Это объясняется тем, что модель учится определять цифры в обучающей выборке. Она еще не видела ни одного изображения из контрольной выборки. О модели, которая очень хорошо справляется с обучающими данными, но плохо с контрольными, говорят, что она «переобучилась».

Один из недостатков нейронных сетей⁴ заключается в том, что они являются своеобразными «черными ящиками». Мы передаем в сеть обучающие данные, оптимизируем веса и применяем модель к контрольным данным. В случае получения высокой оценки появляется соблазн двигаться дальше, не вдаваясь в детали, как алгоритм приходит к своим выводам.

Эта проблема может показаться несущественной, если алгоритм просто читает почтовые индексы, но если он принимает решения, цена ошибки в которых очень высока, понимание особенностей его работы имеет большое значение. Например, представьте, что нейронные сети принимают решения о том, как торговать на фондовом рынке, кто имеет право на получение банковского кредита или насколько высокую страховую ставку следует взимать с клиента.

Чтобы увидеть, как «думает» наш алгоритм, давайте визуализируем матрицу весов, поступающую в каждый нейрон в скрытом слое. Для этого запустим код

¹ <http://bit.ly/2mRDi0l>.

² <http://bit.ly/2G8eSpa>.

³ <http://bit.ly/2rDgnHD>.

⁴ <http://bit.ly/2jSTlbr>.

из примера 3.8. Этот код предназначен для выполнения в Jupyter Notebook и использует `matplotlib`, библиотеку визуализации данных для Python, которую можно установить командой `pip install matplotlib`. Вспомогательная команда `%matplotlib inline` сообщает среде Jupyter Notebook, что вы хотите отобразить изображения в ячейках кода.

Пример 3.8. Визуализация скрытого слоя в нашей нейронной сети

```
# pip install matplotlib
import matplotlib.pyplot as plt

# Для визуализации данных в Jupyter Notebook
%matplotlib inline

fig, axes = plt.subplots(10,10)
fig.set_figwidth(20)
fig.set_figheight(20)

for coef, ax in zip(mlp.coefs_[0].T, axes.ravel()):
    ax.imshow(coef.reshape(8, 8), cmap=plt.cm.gray, interpolation='bicubic')
    ax.set_xticks(())
    ax.set_yticks(())

plt.show()
```

Наш скрытый слой содержит 100 нейронов, которые отвечают за некоторые аспекты, помогающие сети выбрать ту или иную цифру. Входной слой содержит 64 нейрона (потому что входные изображения имеют размеры 8×8 пикселей), поэтому каждый из 100 нейронов в скрытом слое получает на входе 64 значения. В примере 3.8 мы визуализируем матрицы весов во всех нейронах, преобразуя эти 64 значения в матрицу 8×8 пикселей оттенков серого цвета, как показано на рис. 3.5. Сто нейронов в скрытом слое нашей нейронной сети изображены в виде сетки 10×10 .

Большинство изображений на рис. 3.5 выглядят хаотическим нагромождением пикселей, но некоторые из них определенно напоминают цифры, и теперь вы можете видеть, как думает сеть, как она оценивает различные формы и как взвешивает различные значения пикселей, пытаясь распознать цифру.

Наконец, рассмотрим пример 3.9, демонстрирующий использование нашей обученной нейронной сети для распознавания цифр из контрольной выборки. Для этого понадобится библиотека `numpy`, которую можно установить командой `pip install numpy`.

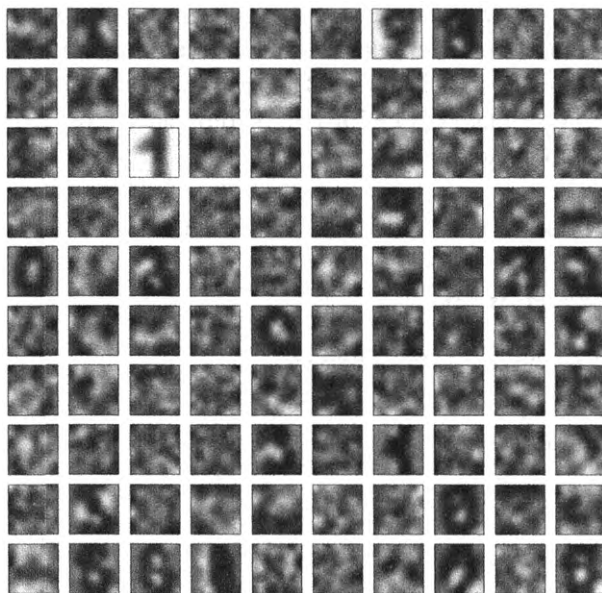


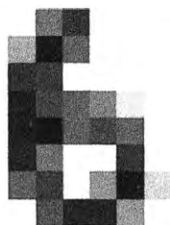
Рис. 3.5. Визуальное представление весовых матриц всех нейронов в скрытом слое

Пример 3.9. Использование обученной нейронной сети для классификации нескольких изображений из контрольной выборки

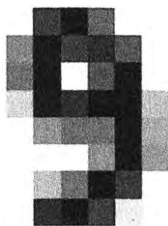
```
import numpy as np # pip install numpy
predicted = mlp.predict(X_test)

for i in range(5):
    image = np.reshape(X_test[i], (8,8))
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    plt.axis('off')
    plt.show()
    print('Ground Truth: {0}'.format(y_test[i]))
    print('Predicted: {0}'.format(predicted[i]))
```

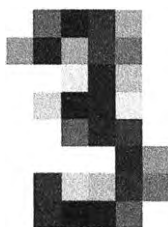
Некоторые результаты, полученные в ходе выполнения примера 3.9, показаны на рис. 3.6. Под каждым изображением выводится истинная цифра, за которой следует цифра, предсказанная классификатором. Как видите, сеть точно классифицировала изображения.



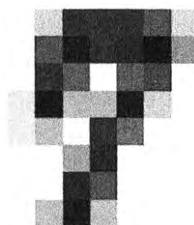
Ground Truth: 6
Predicted: 6



Ground Truth: 9
Predicted: 9



Ground Truth: 3
Predicted: 3



Ground Truth: 7
Predicted: 7

Рис. 3.6. Результаты выполнения примера 3.9, где можно видеть несколько изображений рукописных цифр из контрольной выборки, а также цифры — истинную (Ground Truth) и предсказанную обученной нейронной сетью

3.4.3. Распознавание объектов на фотографиях с помощью предварительно обученных нейросетей

Нейронные сети — в частности, системы «глубокого обучения», которые представляют собой нейронные сети с многими сложными скрытыми слоями, — преобразили целые отрасли. То, что сегодня принято считать «искусственным интеллектом», — это в основном многочисленные приложения систем глубокого обучения, в том числе: машинный перевод, компьютерное зрение, анализ естественного языка, создание текстов на естественном языке и многое другое.

Желающим еще больше приблизиться к пониманию особенностей работы искусственных нейронных сетей хорошим источником знаний послужит книга Майкла Нильсена (Michael Nielsen) *Neural Networks and Deep Learning*¹, свободно доступная в интернете. Также рекомендую прочитать *Machine Learning for Artists*², особенно главу «Looking Inside Neural Networks»³.

Подготовка и обучение искусственной нейронной сети распознаванию объектов — сложная задача. Для ее решения требуется подобрать правильную архитектуру и гиперпараметры нейронной сети, получить большую коллекцию предварительно промаркированных изображений, а затем выделить время для обучения, достаточно продолжительное, чтобы добиться хороших результатов.

К счастью, вся эта работа уже выполнена, и ее результатами может воспользоваться любой, имеющий доступ к современным API компьютерного зрения.

В этом разделе мы используем Google Cloud Vision API — инструмент, позволяющий разработчикам анализировать изображения с использованием мощных нейронных сетей, предварительно обученных в Google. Кроме всего прочего он способен обнаруживать на изображениях самые разные объекты и лица, извлекать текст и определять контент для взрослых.

Перейдите на страницу Cloud Vision API⁴, прокрутите ее вниз до надписи Try the API (Опробовать API). Для опробования вы должны зарегистрировать аккаунт в Google Cloud Platform, если у вас его еще нет. На момент написания

¹ <http://bit.ly/2IzOycW>.

² <http://bit.ly/2rChK9i>.

³ <http://bit.ly/2Kimo3c>.

⁴ <http://bit.ly/2IEmOny>.

этих строк в Google предлагалась бесплатная пробная подписка¹ с кредитом в 300 долларов США на 12 месяцев. При регистрации вам придется указать реквизиты кредитной карты, но они будут использоваться только для идентификации. Если у вас нет кредитной карты, вместо ее реквизитов можно указать реквизиты банковского счета.

Далее вы должны создать проект² в Google Cloud Platform. Его можно считать аналогом клиента, который вы создали в платформе разработчиков Instagram.

Проект можно создать на странице диспетчера облачных ресурсов Cloud Resource Manager³. Создайте его, дав ему любое имя по своему выбору (например, «MTSW»). Свяжите его со своей учетной записью. Даже если вы оформите платную подписку, первые 1000 вызовов Cloud Vision API вы сможете сделать бесплатно.

Создав свой проект, откройте панель инструментов API Dashboard⁴. Она выглядит примерно так, как показано на рис. 3.7.

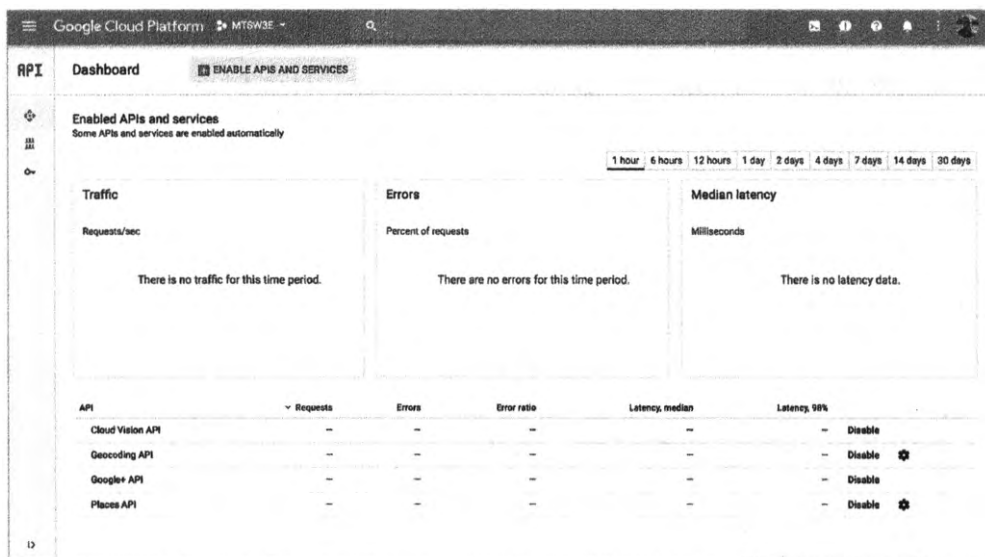


Рис. 3.7. Панель инструментов Google Cloud Platform API Dashboard

¹ <http://bit.ly/2wCXKbC>.

² <http://bit.ly/2rEOZul>.

³ <http://bit.ly/2wyV5zD>.

⁴ <http://bit.ly/2rARWdU>.

Для опробования вам понадобится Cloud Vision API, поэтому подключите его, щелкнув на кнопке **Enable APIs and Services** (Подключить API и службы), затем отыщите Vision API. Подключите API к своему проекту.

Наконец, нужно получить ключ API. В панели инструментов API Dashboard перейдите на страницу **Credentials** (Учетные данные)¹. Щелкните на ссылке **Create credentials** (Создать учетные данные) и выберите **API key** (Ключ API).

Поздравляю! Теперь у вас есть все, что нужно для доступа к Google Cloud Vision API. Процедура, описанная выше, может показаться чересчур сложной, однако все эти шаги предусмотрены компанией Google для защиты своей платформы и предотвращения недобросовестного использования ею технологий.

Скопируйте только что созданный ключ API и вставьте его в пример 3.10, сохранив в переменной `GOOGLE_API_KEY`. Облачная платформа Google Cloud Platform предлагает библиотеку программного доступа к платформе для языка Python. Установите ее, выполнив команду **`pip install google-api-python-client`**. Для обработки изображений мы также будем использовать библиотеку Pillow. Установите ее командой **`pip install Pillow`**.

Пример 3.10. Использование Google Cloud Vision API для классификации изображений

```
import base64
import urllib
import io
import os
import PIL # pip install Pillow
from IPython.display import display, Image

GOOGLE_API_KEY = ''

# pip install google-api-python-client
from googleapiclient.discovery import build
service = build('vision', 'v1', developerKey=GOOGLE_API_KEY)

cat = 'resources/ch05-instagram/cat.jpg'

def label_image(path=None, URL=None, max_results=5):
    '''Читает изображение из файла (находящегося в локальной файловой
    системе или в интернете) и передает данные в Google Cloud Vision API
    для классификации. Ссылка на изображение в интернете должна передаваться
    в именованном аргументе URL, а путь к локальному файлу – в именованном
```

¹ <http://bit.ly/2I9JT1H>.

```
аргументе pass. Для ограничения количества вариантов классификации,
возвращаемых Cloud Vision API, используйте именованный аргумент max_results.
...
if URL is not None:
    image_content = base64.b64encode(urllib.request.urlopen(URL).read())
else:
    image_content = base64.b64encode(open(path, 'rb').read())
service_request = service.images().annotate(body={
    'requests': [{
        'image': {
            'content': image_content.decode('UTF-8')
        },
        'features': [{
            'type': 'LABEL_DETECTION',
            'maxResults': max_results
        }]
    }]
})
labels = service_request.execute()['responses'][0]['labelAnnotations']
if URL is not None:
    display(Image(url=URL))
else:
    display(Image(path))
for label in labels:
    print ('[{0:3.0f}%]: {1}'.format(label['score']*100, label['description']))

return

# Вызов функции классификации изображения с фотографией кошки
label_image(cat)
```

Взаимодействие с Vision API реализуется довольно сложно, поэтому пример 3.10 содержит все, что необходимо для обнаружения и классификации объектов на изображении, которое хранится в локальной файловой системе или в интернете.

Код в примере 3.10 открывает файл с изображением кошки (рис. 3.8), читает его и посылает в Google Vision API. Собственно запрос формируется в переменной `service_request`.

Вот результат выполнения этого кода:

```
[ 99%]: cat
[ 94%]: fauna
[ 93%]: mammal
[ 91%]: small to medium sized cats
[ 90%]: whiskers
```



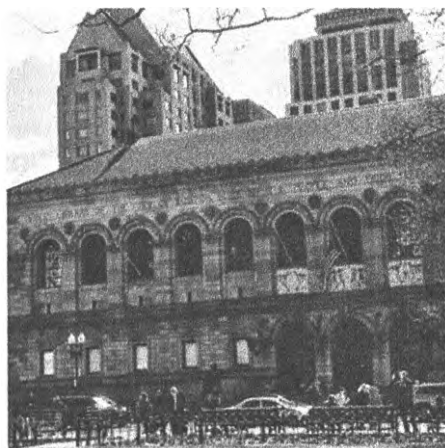
Рис. 3.8. Фотография кошки на снегу (изображение представлено пользователем Von.grzanek на условиях лицензии CC BY-SA 3.0 или GFDL, взято из Wikimedia Commons)

Программный интерфейс Cloud Vision API вернул пять вариантов классификации фотографии, из которых наибольшую вероятность — 99% — получил вариант *cat* (кошка). Другие варианты — *fauna* (фауна), *mammal* (млекопитающее) и остальные — также оказались достаточно точными и уместными.

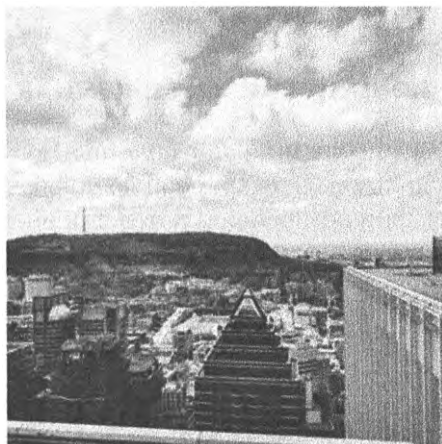
Поэкспериментируйте с этим кодом, отправляя для анализа другие файлы с вашего компьютера, и посмотрите, как они будут классифицированы.

3.5. Применение нейронных сетей для анализа постов в Instagram

Теперь, когда вы познакомились с основами и получили код доступа к мощным нейронным сетям, размещенным в облаке, соберем полную картину из разрозненных фрагментов. В этом разделе рассматривается распознавание объектов и определение лиц, а также используются наши ленты в Instagram, как источники изображений.



[92%]: building
[75%]: medieval architecture
[74%]: classical architecture
[71%]: facade
[61%]: city



[83%]: city
[81%]: sky
[69%]: roof
[67%]: building
[67%]: panorama

Рис. 3.9. Результаты выполнения кода из примера 3.11 с изображениями в ленте Instagram автора

3.5.1. Классификация содержимого изображения

Самое сложное осталось позади. Теперь мы готовы передать изображения из наших лент Instagram в Cloud Vision API. Изображения находятся не на локальном компьютере, а на серверах Instagram. Но не волнуйтесь. Наша функция `label_image` предусматривает возможность передачи URL изображения, что мы делаем в примере 3.11.

Пример 3.11. Распознавание объектов и классификация изображений в вашей ленте Instagram

```
uri = ('https://api.instagram.com/v1/users/self/media/recent/?access_token='
response = requests.get(uri+ACCESS_TOKEN)
recent_posts = response.json()

for post in recent_posts['data']:
    url = post['images']['low_resolution']['url']
    label_image(URL=url)
```

Опробуйте этот пример у себя и посмотрите, насколько точно будет выполнена классификация. На рис. 3.9 показано несколько примеров вывода.

3.5.2. Определение лиц на изображениях

Системы распознавания лиц¹ чрезвычайно полезны. Возможно, вы уже сталкивались с ними, если использовали современную цифровую камеру, например, в смартфоне. В фотографии они используются для идентификации субъекта и фокусировки изображения на субъекте.

Для определения лиц нужно немного изменить запрос. Измененный код приводится в примере 3.12.

Пример 3.12. Пример кода, использующего функцию обнаружения лиц в Google's Cloud Vision API

```
from PIL import Image as PImage
from PIL import ImageDraw

def detect_faces(path=None, URL=None):
    '''Читает изображение из файла (находящегося в локальной файловой
```

¹ <http://bit.ly/2KTvuEy>.

системе или в интернете) и передает данные в Google Cloud Vision API для определения лиц. Ссылка на изображение в интернете должна передаваться в именованном аргументе URL, а путь к локальному файлу - в именованном аргументе path. Для ограничения количества вариантов классификации, возвращаемых Cloud Vision API, используйте именованный аргумент max_results.

```
...
if URL is not None:
    image_content = base64.b64encode(urllib.request.urlopen(URL).read())
else:
    image_content = base64.b64encode(open(path, 'rb').read())
service_request = service.images().annotate(body={
    'requests': [{
        'image': {
            'content': image_content.decode('UTF-8')
        },
        'features': [{
            'type': 'FACE_DETECTION',
            'maxResults': 100
        }]
    }]
})
try:
    faces = service_request.execute()['responses'][0]['faceAnnotations']
except:
    # Лиц не найдено...
    faces = None
if URL is not None:
    im = PImage.open(urllib.request.urlopen(URL))
else:
    im = PImage.open(path)
draw = ImageDraw.Draw(im)

if faces:
    for face in faces:
        box = [(v.get('x', 0.0), v.get('y', 0.0))
               for v in face['fdBoundingPoly']['vertices']]
        draw.line(box + [box[0]], width=5, fill='#ff8888')

display(im)
return
```

Система определения лиц применяется к ленте Instagram так же, как к любому отдельному изображению. Здесь мы применили ее к фотографии группы «Битлз», сделанной в 1967 году. Как видно на рис. 3.10, алгоритм правильно идентифицировал все четыре лица.



Рис. 3.10. Результат определения лиц кодом из примера 3.12 на фотографии группы «Битлз», сделанной в 1967 году (оригинальное изображение предоставлено Parlophone Music Sweden на условиях лицензии CC BY 3.0, взято из Wikimedia Commons)

3.6. Заключительные замечания

Компания Instagram создала чрезвычайно популярную платформу для обмена фотографиями, которой пользуются сотни миллионов людей по всему миру. Данные, размещаемые в Instagram, отличаются от данных, которые анализировали в других главах. Основным контентом в Instagram являются изображения с подписями, поэтому для его анализа требуется другой набор инструментов.

Одними из самых передовых инструментов анализа изображений, существующих ныне, являются искусственные нейронные сети, обученные распознаванию предметов. Под словом «обученные» подразумевается, что эти нейронные сети «видели» тысячи примеров, классифицированных людьми, и настроены для распознавания тех же ассоциаций.

Системы компьютерного зрения этого типа встраиваются в камеры, в системы безопасности, беспилотные автомобили и множество других продуктов. В этой главе мы показали путь к созданию собственных систем компьютерного зрения и рассказали, как подключиться к некоторым из самых мощных API.

Исследователи, разрабатывающие эти системы, часто должны платить людям-помощникам за просмотр больших баз данных изображений и классификацию каждого изображения вручную. Это трудоемко и дорого. Социальные платформы, такие как Facebook и Instagram, куда каждый день люди выгружают бесчисленное количество изображений и добавляют свои хештеги, понимают, насколько полезна эта информация. В Facebook приступили к использованию миллиардов изображений с хештегами из Instagram, чтобы обучить свои системы искусственного интеллекта¹ воспринимать фотографии подобно людям.

Обучая машины визуальному восприятию мира, мы открываем невероятные возможности. Наиболее важными результатами этих исследований являются технологии, помогающие людям, такие как беспилотные транспортные средства, способные безопасно перемещаться среди пешеходов, велосипедистов и других препятствий, или методы медицинской визуализации, которые могут точно идентифицировать опухоль на самых ранних стадиях. Конечно, эти инструменты можно использовать и для притеснения. Знание, как работают эти технологии, является первым шагом к пропаганде их надлежащего использования.



Исходный код примеров для этой и всех других глав доступен на GitHub² в удобном формате Jupyter Notebook, который вы можете опробовать, не покидая веб-браузера.

3.7. Упражнения

- Instagram хранит выгруженные фотографии в разных разрешениях, и вы сможете найти их URL в метаданных, возвращаемых API вместе с сообщением. Сравните точность классификации изображения или распознавания лиц на одном и том же изображении с разным разрешением.

¹ <http://bit.ly/2IzMPes>.

² <http://bit.ly/Mining-the-Social-Web-3E>.

- Попробуйте поэкспериментировать с архитектурой нейронной сети, созданной нами для идентификации рукописных цифр. В этой главе мы использовали один скрытый слой со 100 нейронами. Как изменится точность на контрольной выборке при уменьшении числа нейронов в скрытом слое до 50, 20, 10 или 5? А как скажется на точности увеличение числа скрытых слоев? Прочитайте описание классификатора MLP¹ в документации к библиотеке `scikit-learn`. Аббревиатура MLP расшифровывается как `multilayer perceptron`² (многослойный перцептрон) — это разновидность архитектуры нейронных сетей, которую мы использовали.
- Когда фотографии выгружаются в Instagram с телефона, они обычно привязываются к текущим географическим координатам. Для определения местоположения во время привязки фотографий Instagram использует датчик GPS телефона (если вы разрешили приложению доступ к службе определения местоположения) и дает возможность отметить местоположение при редактировании публикации. Посмотрите, сможете ли вы найти информацию с координатами в метаданных записи. Обойдите в цикле посты в своей ленте и выведите на экран все значения широты и долготы.
- В главе 4 вы познакомитесь с еще одним программным интерфейсом Google — службой геокодирования, которая используется для поиска местоположений на Земле. Закончив читать ее, вернитесь к примерам анализа данных из Instagram и попробуйте извлечь географическую информацию из метаданных сообщений Instagram или найти объект, упоминаемый в описании к изображению. Попробуйте создать KML-файл на основе этих данных и откройте его в Google Earth. Это самый простой способ увидеть на карте места, полученные из фотографии.
- Если вы добавляете хештеги при публикации фотографий в Instagram, сравните их с метками, которые возвращает Google Cloud Vision API для ваших фотографий. Как бы вы использовали систему компьютерного зрения, такую как Cloud Vision API, для создания автоматизированной системы рекомендаций по выбору хештегов?

3.8. Онлайн-ресурсы

Ниже приводится список ссылок, упоминавшихся в этой главе, которые могут оказаться полезными для вас:

¹ <http://bit.ly/2jQt90V>.

² <http://bit.ly/2Ie80ME>.

- Документация разработчика Instagram¹.
- Безопасное окружение для приложений Instagram².
- Передача репрезентативного состояния (REST)³.
- Instagram Graph API⁴.
- Конечные точки Instagram API⁵.
- Сеть доставки содержимого (CDN)⁶.
- Биологические нейронные сети⁷.
- Искусственные нейронные сети⁸.
- Алгоритм обратного распространения ошибки⁹.
- Функция потерь¹⁰.
- Компьютерное зрение¹¹.
- Определение лиц¹².
- Оптическое распознавание символов (OCR)¹³.
- База данных MNIST¹⁴.
- База данных EMNIST¹⁵.
- *Hands-On Machine Learning with Scikit-Learn and TensorFlow*¹⁶.

¹ <http://bit.ly/2Ibb4JL>.

² <http://bit.ly/2Ia88Nr>.

³ <http://bit.ly/2rC8oJW>.

⁴ <http://bit.ly/2jTGHce>.

⁵ <http://bit.ly/2I9iAEF>.

⁶ <http://bit.ly/2Gb0DzH>.

⁷ <http://bit.ly/2KWaOvR>.

⁸ <http://bit.ly/2IcVrkK>.

⁹ <http://bit.ly/2jRgYRB>.

¹⁰ <http://bit.ly/2KdyVoo>.

¹¹ <http://bit.ly/2IygU79>.

¹² <http://bit.ly/2KTvuEy>.

¹³ <http://bit.ly/2IjnqLX>.

¹⁴ <http://bit.ly/2IaAxmC>.

¹⁵ <http://bit.ly/2rAZrS4>.

¹⁶ <http://oriel.ly/2KV4XS> (Жерон О. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. Концепции, инструменты и техники для создания интеллектуальных систем. М.: Вильямс, 2018. — Примеч. пер.)

- *Neural Networks and Deep Learning*, глава 1 («Using Neural Nets to Recognize Handwritten Digits»)¹.
- Многослойный перцептрон².
- Классификатор MLP в библиотеке `scikit-learn`³.
- Сверточные нейронные сети⁴.
- Примеры лучших алгоритмов компьютерного зрения для решения задач классификации, включая MNIST⁵.
- Статья «The Dark Secret at the Heart of AI»⁶.
- *Machine Learning for Artists*⁷.
- Глава «Looking inside neural nets» (из книги *Machine Learning for Artists*)⁸.
- Google Vision API⁹.
- Google Cloud Platform Console¹⁰.
- Google Cloud Platform API Dashboard¹¹.

¹ <http://bit.ly/2IzOycW>.

² <http://bit.ly/2Ie80ME>.

³ <http://bit.ly/2jQt90V>.

⁴ <http://bit.ly/2rDgnHD>.

⁵ <http://bit.ly/2G8eSpa>.

⁶ <http://bit.ly/2jSTlbr>.

⁷ <http://bit.ly/2rChK9i>.

⁸ <http://bit.ly/2Kimo3c>.

⁹ <http://bit.ly/2IEmOny>.

¹⁰ <http://bit.ly/2Kin4FM>.

¹¹ <http://bit.ly/2rARWdU>.

4

LinkedIn: классификация по профессиям, группировка коллег и многое другое

В этой главе описываются методы и особенности анализа данных, хранящихся в LinkedIn — социальной сети, ориентированной на профессиональные и деловые отношения. На первый взгляд LinkedIn может показаться похожей на любую другую социальную сеть, но данные, возвращаемые ее программным интерфейсом, имеют совершенно иную природу. Если Twitter можно сравнить с многолюдным форумом, напоминающим городскую площадь, а Facebook — с очень большой комнатой, наполненной друзьями и родственниками, ведущими разговоры, которые (обычно) уместны за ужином, то LinkedIn можно сравнить с частным мероприятием с полуоформальным дресс-кодом, где все стремятся соблюсти хорошие манеры и передать знания и опыт, приобретенные на профессиональном поприще.

Учитывая особенно чувствительный характер данных, спрятанных в LinkedIn, программный интерфейс этой социальной сети имеет свои нюансы, делающие его несколько отличным от многих других программных интерфейсов, рассматривающихся в этой книге. Люди, присоединившиеся к LinkedIn, в основном заинтересованы в расширении сфер деятельности, которые они предоставляют, а не в произвольном общении, и обязательно предоставляют сведения о деловых отношениях, послужном списке и многом другом. Например, в LinkedIn можно получить доступ ко всем сведениям о *ваших* контактах, их образовании и предыдущих рабочих местах, но нельзя определить, связаны ли два произвольных человека. Эта информация не поддерживается намеренно. LinkedIn API нельзя смоделировать в виде социального графа, как Facebook или Twitter, поэтому он требует, чтобы вы задавали разные типы вопросов о доступных вам данных.

Оставшаяся часть этой главы посвящена настройке доступа к данным с помощью LinkedIn API и знакомит с некоторыми базовыми методами анализа

данных, которые помогут вам найти коллег в соответствии с заданной мерой сходства и ответить на следующие вопросы:

- Какие из ваших контактов наиболее похожи по такому критерию, как название профессии?
- Кто из ваших контактов работал в компании, где вы хотите получить работу?
- В каком географическом регионе проживает большинство ваших контактов?

Во всех случаях анализ выполняется методом кластеризации, фактически следуя одному и тому же шаблону: извлечь некоторые признаки из профиля контакта, определить меру сходства для сравнения признаков из каждого профиля и использовать метод кластеризации для объединения контактов в группы «с наибольшим сходством». Этот подход хорошо работает применительно к данным из LinkedIn, и вы можете использовать его в отношении практически любых других данных, с которыми вам доведется столкнуться.



Загрузите последний исправленный исходный код примеров для этой (и любой другой) главы, доступный по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Воспользуйтесь также преимуществами виртуальной машины, описанными в приложении А, чтобы получить максимальное удовольствие от опробования примеров кода.

4.1. Обзор

Эта глава знакомит со сведениями, составляющими основу машинного обучения и в целом немного более продвинутыми, чем две предыдущие главы. Прежде чем приступить к этой главе, рекомендуется внимательно прочитать две предыдущие. В этой главе вы узнаете о:

- платформе разработчиков LinkedIn и способах выполнения запросов к API;
- трех основных типах кластеризации, лежащих в основе машинного обучения, которые могут найти применение практически в любой предметной области;
- приемах очистки и нормализации данных;
- геокодировании — способе получения набора координат из текстового описания местоположения;

- визуализации географических данных с помощью Google Earth и картограмм.

4.2. LinkedIn API

Чтобы повторить примеры из этой главы, вам понадобится аккаунт в LinkedIn и несколько контактов в вашей профессиональной сети. Если у вас нет аккаунта в LinkedIn, вы все еще сможете применить основные методы кластеризации, о которых узнаете, в других областях, но эта глава уже не будет столь привлекательной из-за невозможности опробовать примеры без своих данных в LinkedIn. Начните развивать свою сеть LinkedIn, если у вас ее еще нет, чтобы сделать ценные инвестиции в вашу профессиональную жизнь.

Несмотря на то что в большинстве случаев в этой главе для анализа используется файл контактов LinkedIn (в формате CSV), который можно загрузить, этот раздел продолжает традицию, заложенную в других главах, и предлагает обзор LinkedIn API. Если организация LinkedIn API вас не интересует и вы предпочли бы сразу заняться анализом, то переходите к разделу «Загрузка файла с информацией о контактах в LinkedIn» и возвращайтесь сюда, когда у вас появится желание познакомиться с деталями выполнения запросов к API.

4.2.1. Выполнение запросов к LinkedIn API

Как и в случае с другими социальными сетями, такими как Twitter и Facebook (обсуждались в предыдущих главах), первый шаг для получения доступа к LinkedIn API заключается в создании приложения. Вы можете создать пример приложения, посетив портал разработчика¹; здесь вы должны скопировать и сохранить идентификатор и секрет клиента вашего приложения — это ваши учетные данные для аутентификации, которые вы будете использовать для программного доступа к API. На рис. 4.1 показана форма, которую вы увидите после создания приложения.

Для получения доступа к вашим личным данным посредством API вам останется только передать эти учетные данные библиотеке, которая позаботится обо всем остальном. Если вы решили не пользоваться виртуальной машиной, сопровождающей книгу, вам нужно будет установить эту библиотеку, выполнив команду `pip install python3-linkedin` в терминале.

¹ <https://bit.ly/2swubEU>.



Дополнительные сведения об использовании протокола OAuth 2.0, которые понадобятся при создании приложения, требующего авторизации произвольного пользователя для доступа к данным учетной записи, вы найдете в приложении Б.

Authentication Keys

Client ID:

Client Secret:

Default Application Permissions

☒ r_basicprofile ☐ r_emailaddress ☐ rw_company_admin
☐ w_share

OAuth 2.0

Authorized Redirect URLs:

OAuth 1.0a

Default "Accept" Redirect URL:

Default "Cancel" Redirect URL:

Рис. 4.1. Чтобы получить доступ к LinkedIn API, создайте приложение, а затем скопируйте и сохраните идентификатор и секрет клиента (здесь они затерты), указанные на странице со сведениями о приложении

В примере 4.1 демонстрируется сценарий, использующий учетные данные LinkedIn для создания экземпляра класса `LinkedInApplication`, который обеспечивает доступ к данным из учетной записи. Обратите внимание на последнюю

строку сценария — она извлекает базовую информацию о профиле, включая имя и заголовок. Прежде чем двинуться дальше, воспользуйтесь моментом и прочитайте в документации с описанием REST API¹, где предлагается широкий обзор возможностей, какие операции LinkedIn API доступны вам как разработчику. Несмотря на то что мы будем обращаться к API с помощью пакета для Python, который скрывает от нас выполняемые HTTP-запросы, документация с описанием API послужит вам авторитетным руководством, к тому же большинство популярных библиотек имитируют его стиль.

Пример 4.1. Использование учетных данных OAuth для аутентификации в LinkedIn и получения токена доступа, необходимого для извлечения данных из вашего профиля

```
from linkedin import linkedin # pip install python3-linkedin

APPLICATION_KEY = ''
APPLICATION_SECRET = ''

# Адрес URL для переадресации должен совпадать с URL, указанным
# в настройках приложения
RETURN_URL = 'http://localhost:8888'

authentication = linkedin.LinkedInAuthentication(
    APPLICATION_KEY,
    APPLICATION_SECRET,
    RETURN_URL)

# Откройте этот URL в браузере и скопируйте фрагмент после 'code='
print(authentication.authorization_url)

# Вставьте его сюда, но не включайте '&state=' и все, что следует далее
authentication.authorization_code = ''

result = authentication.get_access_token()

print ("Access Token:", result.access_token)
print ("Expires in (seconds):", result.expires_in)

# Передать токен доступа в приложение
app = linkedin.LinkedInApplication(token=result.access_token)

# Извлечь информацию из профиля
app.get_profile(selectors=['id', 'first-name', 'last-name',
                          'location', 'num-connections', 'headline'])
```

¹ <http://linkd.in/1a1lZuj>.

Через экземпляр `LinkedInApplication` можно выполнять те же вызовы, что и через REST API. В документации¹ на GitHub с описанием `python-linkedin` вы найдете несколько примеров запросов, которые помогут вам начать. Особый интерес представляют `Connections API` и `Search API`. Как говорилось выше во вступительном обсуждении, вы не сможете получить список «друзей ваших друзей» (или «контакты ваших контактов», если выразаться на языке LinkedIn), но `Connections API` позволяет получить список ваших контактов и служит отправной точкой для получения информации из профиля. `Search API` предлагает средства для поиска людей, компаний или вакансий, доступных в LinkedIn.

Существуют также другие API, и вам определенно стоит уделить некоторое время для знакомства с ними. Однако следует отметить, что на протяжении многих лет в LinkedIn постоянно вносились изменения в API, ограничивающие доступ к информации. Например, попытка получить все данные о ваших контактах через API может завершиться ошибкой 403 («Запрещено»). С другой стороны, LinkedIn по-прежнему позволяет загрузить архив с информацией обо всех ваших контактах, о чем мы поговорим в разделе «Загрузка файла с информацией о контактах в LinkedIn». Этот архив содержит те же данные, доступные аутентифицированному пользователю на веб-сайте LinkedIn.



Будьте осторожны, экспериментируя с LinkedIn API: ограничения на количество запросов сбрасываются в полночь по Гринвичу, из-за чего один цикл отладки может полностью нарушить ваши планы на следующие 24 часа.

Пример 4.2 демонстрирует, как получить послужной список из вашего собственного профиля.

Пример 4.2. Вывод послужного списка из вашего профиля и профилей контактов

```
import json
```

```
# Перечень всех селекторов полей, которые можно указывать для получения  
# дополнительной информации, вы найдете на странице:  
# https://developer.linkedin.com/docs/fields/positions.
```

```
# Вывести свой послужной список...
```

```
my_positions = app.get_profile(selectors=['positions'])  
print(json.dumps(my_positions, indent=1))
```

¹ <http://bit.ly/1a1m2Gk>.

В возвращаемых результатах можно заметить ряд интересных сведений о каждой позиции, включая название компании, отрасль, краткое описание выполняемой работы и дату трудоустройства:

```
{
  "positions": {
    "_total": 10,
    "values": [
      {
        "startDate": {
          "year": 2013,
          "month": 2
        },
        "title": "Chief Technology Officer",
        "company": {
          "industry": "Computer Software",
          "name": "Digital Reasoning Systems"
        },
        "summary": "I lead strategic technology efforts...",
        "isCurrent": true,
        "id": 370675000
      },
      {
        "startDate": {
          "year": 2009,
          "month": 10
        }
      },
      ...
    ]
  }
}
```

Как нетрудно догадаться, некоторые ответы API могут содержать не всю информацию, которую вы хотели бы получить, а некоторые — больше, чем вам нужно. Чтобы не выполнять несколько вызовов API подряд с последующим объединением сведений или не удалять избыточную информацию, которую вы не хотите хранить, можно воспользоваться синтаксисом селекторов полей¹, чтобы явно указать, какая информация должна быть возвращена в ответе. В примере 4.3 показано, как получить только поля `name`, `industry` и `id` в ответ на запрос послужного списка.

¹ <http://bit.ly/2E7vahT>.

Пример 4.3. Использование синтаксиса селекторов полей для передачи в API уточняющей информации о запросе

```
# Дополнительную информацию о синтаксисе селекторов полей ищите на странице:  
# http://bit.ly/2E7vahT
```

```
my_positions = app.get_profile(selectors=['positions:(company:(name,industry,id))'])  
print json.dumps(my_positions, indent=1)
```

Теперь, когда вы познакомились с основными доступными API, получили несколько ссылок на документацию и попробовали выполнить несколько вызовов, можно приступать к работе с LinkedIn.

4.2.2. Загрузка файла с информацией о контактах в LinkedIn

Программный интерфейс открывает доступ ко многим аспектам, которые доступны вам, как аутентифицированному пользователю, в вашем профиле на сайте <http://linkedin.com>. Однако те же самые сведения, которые вам понадобятся в большей части этой главы, можно получить, экспортировав свои контакты LinkedIn в файл CSV. Для этого перейдите на страницу **Settings & Privacy** (Настройки и конфиденциальность) в LinkedIn и найдите ссылку **Download your data** (Загрузить данные) или непосредственно перейдите к диалогу **Export LinkedIn Connections** (Экспортировать контакты LinkedIn), изображенному на рис. 4.2.

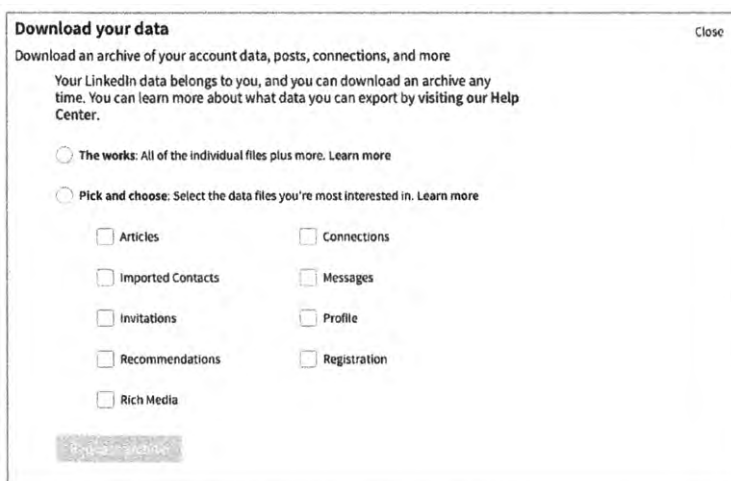


Рис. 4.2. Малоизвестная особенность LinkedIn — возможность экспортировать все свои контакты в удобном и переносимом формате CSV

4.3. Краткое введение в приемы кластеризации данных

Теперь, получив представление о том, как получить доступ к LinkedIn API, перейдем к конкретному анализу и подробно обсудим *кластеризацию*¹ — метод машинного обучения без учителя, считающийся основным в любом наборе инструментов анализа данных. Алгоритм кластеризации принимает коллекцию элементов и делит их на более мелкие коллекции (кластеры) согласно некоторому критерию, предназначенному для сравнения элементов в коллекции.



Кластеризация — это фундаментальный метод анализа данных, поэтому, чтобы вы могли получить более полное представление о нем, эта глава включает сноски и примечания с описанием математического аппарата, лежащего в его основе. Хорошо, если вы постараетесь понять эти детали, но, чтобы успешно использовать методы кластеризации, не требуется понимать все тонкости, и, конечно же, от вас не требуется, чтобы вы разобрались в них с первого раза. Вам может потребоваться немного поразмышлять, чтобы переварить некоторые сведения, особенно если у вас нет математической подготовки.

Например, если вы подумываете о переезде в другой город, вы можете попробовать объединить контакты в LinkedIn по географическим регионам, чтобы лучше оценить имеющиеся экономические возможности. Мы вернемся к этой идее чуть позже, а пока вкратце обсудим некоторые нюансы, связанные с кластеризацией.

При реализации решений задач кластеризации данных из LinkedIn или из других источников вы неоднократно будете сталкиваться, по крайней мере, с двумя основными темами (обсуждение третьей приводится во врезке «Роль уменьшения размерности в кластеризации» ниже).

Нормализация данных

Даже при использовании очень хорошего API данные редко бывают представлены в нужном вам формате, — часто требуется нечто большее, чем простое преобразование, чтобы привести данные в форму, пригодную для анализа. Например, пользователи LinkedIn допускают определенные вольности, описывая свои должности, поэтому не всегда удастся получить идеально нормализованные описания. Один руководитель может выбрать название

¹ Этот вид анализа также часто называют методом *приближенного совпадения, нечеткого соответствия* и/или *дедупликацией*.

«главный технический директор», другой — более двусмысленное название «ГТО», а третий может описать ту же должность как-то иначе. Чуть ниже мы вернемся к проблеме нормализации данных и реализуем шаблон для обработки определенных ее аспектов в данных LinkedIn.

Определение сходства

Имея набор хорошо нормализованных элементов, вы можете пожелать оценить сходство любых двух из них, будь то названия должностей или компаний, описание профессиональных интересов, географические названия или любые другие поля, значения которых могут быть представлены произвольным текстом. Для этого вам нужно определить эвристику, оценивающую сходство двух любых значений. В некоторых ситуациях определение сходства вполне очевидно, но в других может быть сопряжено с некоторыми сложностями.

Например, сравнение общего трудового стажа двух человек реализуется простыми операциями сложения, но сравнение более широких профессиональных характеристик, таких как «лидерские способности», полностью автоматизированным образом может оказаться довольно сложной задачей.

РОЛЬ УМЕНЬШЕНИЯ РАЗМЕРНОСТИ В КЛАСТЕРИЗАЦИИ

Нормализация данных и определение сходства — это две главные темы, с которыми вы будете сталкиваться в кластеризации на абстрактном уровне. Но есть еще третья тема — сокращение размерности, которая становится актуальной, как только масштаб данных перестает быть тривиальным. Для группировки элементов в множестве с использованием метрики сходства в идеале желательно сравнить каждый элемент с каждым другим элементом. В этом случае, при наихудшем развитии событий, для множества из n элементов вам придется вычислить степень сходства примерно n^2 раз, чтобы сравнить каждый из n элементов с $n-1$ другими элементами.

В информатике эту ситуацию называют *проблемой квадратичной сложности* и обычно обозначают как $O(n^2)$; в разговорах ее обычно называют «проблемой квадратичного роста большого O ». Проблемы $O(n^2)$ становятся неразрешимыми для очень больших значений n , и в большинстве случаев термин *неразрешимые* означает, что вам придется ждать «слишком долго», пока решение будет вычислено. «Слишком долго» — это могут быть минуты, годы или эпохи, в зависимости от характера задачи и ее ограничений.

Обзор методов уменьшения размерности выходит за рамки текущего рассмотрения, поэтому отметим лишь, что типичный метод уменьшения размерности предусматривает использование функции для организации «достаточно похожих» элементов в фиксированное число групп, чтобы элементы в каждой группе можно было в полной мере считать похожими. Уменьшение размерности часто является не только наукой, но и искусством, и обычно считается конфиденциальной информацией или коммерческой тайной организациями, которые успешно используют ее для получения конкурентного преимущества.

Методы кластеризации являются основной частью арсенала инструментов любого специалиста по анализу данных, потому что почти в любой отрасли — от военной разведки до банковского дела и ландшафтного дизайна — может потребоваться проанализировать по-настоящему огромный объем нестандартных реляционных данных, и рост числа вакансий специалистов по данным за предыдущие годы служит тому явным свидетельством.

Как правило, для сбора какой-либо информации компания создает базу данных, но не каждое поле может содержать значения из некоторого предопределенного набора. Это может быть обусловлено не до конца продуманной логикой работы пользовательского интерфейса приложения, невозможностью заранее определить все допустимые значения или необходимостью дать пользователям возможность вводить любой текст по своему желанию. Как бы то ни было, результат всегда одинаков: вы получаете большой объем нестандартизованных данных. Даже если в определенном поле в общей сложности может храниться N разных строковых значений, некоторое их количество фактически будет обозначать одно и то же понятие. Дубликаты могут возникать по разным причинам — из-за орфографических ошибок, использования аббревиатур или сокращений, а также разных регистров символов.

Как упоминалось выше, это классическая ситуация, возникающая при анализе данных из LinkedIn: пользователи могут вводить свою информацию в свободном текстовом виде, что неизбежно приводит к нарастанию вариаций. Например, если вы решите исследовать свою профессиональную сеть и определить, где работает большинство ваших контактов, вам придется рассмотреть часто используемые варианты написания названий компаний. Даже самые простые названия компаний могут иметь несколько вариантов, с которыми вы почти наверняка столкнетесь (например, «Google» — сокращенная форма «Google, Inc.»), и вам придется учесть все эти варианты, чтобы привести их к стандартной форме. При стандартизации названий компаний хорошей отправной точкой может стать нормализация сокращений в названиях, таких как LLC и Inc.

4.3.1. Нормализация данных для анализа

В качестве необходимого и полезного вступления перед изучением алгоритмов кластеризации рассмотрим несколько типичных ситуаций, с которыми можно столкнуться, решая задачу нормализации данных из LinkedIn. В этом разделе мы реализуем типовой шаблон нормализации названий компаний и должностей. В качестве более продвинутого упражнения мы также кратко обсудим про-

блему устранения неоднозначности и геокодирования географических названий из профиля LinkedIn. (То есть мы попытаемся преобразовать географические названия из профилей LinkedIn, такие как «Greater Nashville Area» (Большой Нэшвилл), в координаты, которые можно нанести на карту.)



Главным результатом усилий по нормализации данных является возможность учитывать и анализировать важные признаки и использовать передовые методы анализа, такие как кластеризация. В случае с данными из LinkedIn мы будем изучать такие признаки, как должности и географические местоположения.

Нормализация и подсчет компаний

Давайте попробуем стандартизировать названия компаний из вашей профессиональной сети. Как рассказывалось выше, извлечь данные из LinkedIn можно двумя основными способами: программным, с помощью LinkedIn API, или с использованием механизма экспортирования профессиональной сети в виде адресной книги, которая включает такие основные сведения, как имя, должность, компания и контактная информация.

Представим, что у нас уже есть CSV-файл с контактами, экспортированный из LinkedIn, и теперь мы можем нормализовать и вывести выбранные сущности, как показано в примере 4.4.



Как описывается в комментариях внутри примеров, вам нужно переименовать CSV-файл с контактами, который вы экспортировали из LinkedIn, следуя инструкциям в разделе «Загрузка файла с информацией о контактах в LinkedIn», и скопировать в определенный каталог, где его сможет найти программный код.

Пример 4.4. Простая нормализация сокращений в названиях компаний

```
import os
import csv
from collections import Counter
from operator import itemgetter
from prettytable import PrettyTable

# Загрузите данные из LinkedIn: https://www.linkedin.com/psettings/member-data.
# Получив запрос, LinkedIn подготовит архив с данными из вашего профиля,
# который вы сможете загрузить. Поместите этот архив в папку
# resources/ch03-linkedin/.

CSV_FILE = os.path.join("resources", "ch03-linkedin", 'Connections.csv')
```

```
# Определить набор преобразований, превращающих первый элемент
# во второй. Здесь перечисляются некоторые распространенные
# сокращения и реализуется их удаление.

transforms = [(' Inc.', ''), (' Inc', ''), (' LLC', ''), (' LLP', ''),
              (' LLC', ''), (' Inc.', ''), (' Inc', '')]

companies = [c['Company'].strip() for c in contacts if c['Company'].strip() != '']

for i, _ in enumerate(companies):
    for transform in transforms:
        companies[i] = companies[i].replace(*transform)

pt = PrettyTable(field_names=['Company', 'Freq'])
pt.align = 'l'
c = Counter(companies)

[pt.add_row([company, freq]) for (company, freq) in
 sorted(c.items(), key=itemgetter(1), reverse=True) if freq > 1]

print(pt)
```

Ниже приводятся результаты простого частотного анализа:

Company	Freq
Digital Reasoning Systems	31
O'Reilly Media	19
Google	18
Novetta Solutions	9
Mozilla Corporation	9
Booz Allen Hamilton	8
...	...



В языке Python поддерживается возможность передачи аргументов функциям путем *разыменования* списка и/или словаря, что иногда очень удобно, как показано в примере 4.4. Например, вызов `f(*args, **kw)` эквивалентен вызову `f(1, 7, x=23)`, где `args` аргументов определяется как список `[1, 7]` и `kw` — как словарь `{'x': 23}`. Другие советы по программированию на языке Python вы найдете в приложении В.

Имейте в виду, что для обработки более сложных ситуаций, например, для нормализации разных названий одной и той же компании, менявшихся с течением времени, таких как O'Reilly Media, вам потребуется написать более замысло-

ватый код. В данном случае название этой компании может быть представлено как O'Reilly & Associates, O'Reilly Media, O'Reilly, Inc. или просто O'Reilly.¹

Нормализация и подсчет должностей

Как нетрудно догадаться, та же проблема нормализации возникает с названиями должностей, с той лишь разницей, что она может стать намного более запутанной, потому что названия должностей могут иметь намного больше вариантов. В табл. 4.1 перечислены некоторые должности, которые можно встретить в компании, занимающейся разработкой программного обеспечения, включая определенное количество естественных вариантов. Сколько разных должностей вы видите в 10 разных названиях?

Таблица 4.1. Примеры названий должностей в технологической отрасли

Название должности
Chief Executive Officer (главный исполнительный директор)
President/CEO (президент/главный исполнительный директор)
President & CEO (президент и главный исполнительный директор)
CEO (главный исполнительный директор)
Developer (разработчик)
Software Developer (разработчик программного обеспечения)
Software Engineer (инженер-программист)
Chief Technical Officer (главный технический директор)
President (президент)
Senior Software Engineer (старший инженер-программист)

Можно, конечно, определить список псевдонимов или сокращений, который приравнивает CEO и Chief Executive Officer, но было бы нецелесообразно вручную определять списки, приравнивающие такие названия, как Software Engineer и Developer во всех возможных областях. Тем не менее, даже в самом худшем случае будет не слишком сложно реализовать решение, которое ужи-

¹ Если вам покажется, что вас поджидают большие сложности, просто представьте, какую работу пришлось проделать специалистам компании Dun & Bradstreet (<http://bit.ly/1a1m40m>), специализирующейся на каталогизации информации и столкнувшейся с задачей составления и сопровождении реестра с названиями компаний на самых разных языках мира.

мает данные до такой степени, чтобы их мог проверить эксперт, и передать эти данные обратно в программу, способную применить их, как это сделал бы эксперт. Такой подход часто выбирается во многих организациях, поскольку позволяет включить цикл контроля качества.

Напомним еще раз, что при работе с любыми наборами данных одной из самых очевидных отправных точек является подсчет, и данная ситуация ничем не отличается. Давайте еще раз воспользуемся теми же идеями, что и при нормализации названий компаний, и реализуем шаблон для нормализации названий должностей, а затем выполним простой частотный анализ этих названий, чтобы заложить основу для последующей кластеризации. При наличии достаточного количества экспортированных контактов можно столкнуться с довольно удивительными вариациями в названиях должностей, но прежде чем начать, рассмотрим пример кода, определяющего некоторые шаблоны для нормализации данных и возвращающего результаты, отсортированные по частоте.

Код в примере 4.5 исследует названия должностей и выводит частоты для них самих и отдельных слов, встречающихся в этих названиях.

Пример 4.5. Стандартизация названий должностей и подсчет их частот

```
import os
import csv
from operator import itemgetter
from collections import Counter
from prettytable import PrettyTable

# Укажите здесь путь к вашему файлу 'Connections.csv'
CSV_FILE = os.path.join('resources', 'ch03-linkedin', 'Connections.csv')

csvReader = csv.DictReader(open(CSV_FILE), delimiter=',', quotechar='"')
contacts = [row for row in csvReader]

transforms = [
    ('Sr.', 'Senior'),
    ('Sr', 'Senior'),
    ('Jr.', 'Junior'),
    ('Jr', 'Junior'),
    ('CEO', 'Chief Executive Officer'),
    ('COO', 'Chief Operating Officer'),
    ('CTO', 'Chief Technology Officer'),
    ('CFO', 'Chief Finance Officer'),
    ('VP', 'Vice President'),
]
```

```
# Прочитать названия в список и разбить на компоненты
# любые составные названия, такие как "President/CEO".
# Аналогично можно обработать другие составные названия,
# такие как "President & CEO", "President and CEO" и т. д.

titles = []
for contact in contacts:
    titles.extend([t.strip() for t in contact['Position'].split('/')
                  if contact['Position'].strip() != ''])

# Заменить распространенные/известные сокращения

for i, _ in enumerate(titles):
    for transform in transforms:
        titles[i] = titles[i].replace(*transform)

# Вывести таблицу названий, отсортировав по частоте

pt = PrettyTable(field_names=['Job Title', 'Freq'])
pt.align = 'l'
c = Counter(titles)
[pt.add_row([title, freq])
 for (title, freq) in sorted(c.items(), key=itemgetter(1), reverse=True)
  if freq > 1]
print(pt)

# Вывести таблицу слов, отсортировав по частоте
tokens = []
for title in titles:
    tokens.extend([t.strip(',') for t in title.split()])
pt = PrettyTable(field_names=['Token', 'Freq'])
pt.align = 'l'
c = Counter(tokens)
[pt.add_row([token, freq])
 for (token, freq) in sorted(c.items(), key=itemgetter(1), reverse=True)
  if freq > 1 and len(token) > 2]
print(pt)
```

Код из примера 4.5 читает записи из файла CSV и пытается нормализовать их, разбивая названия должностей на составляющие по символу слеша (как в названии «President/CEO») и замещая известные сокращения. После этого он выводит результаты частотного анализа полных названий и компонентов, составляющих их.

Это не все отличия от предыдущего упражнения с названиями компаний, тем не менее данный пример служит хорошим начальным шаблоном и дает некоторое представление о возможностях анализа данных.

Ниже приводится пример полученных результатов:

Title	Freq
Chief Executive Officer	19
Senior Software Engineer	17
President	12
Founder	9
...	...

Token	Freq
Engineer	43
Chief	43
Senior	42
Officer	37
...	...

Самое примечательное здесь то, что в результатах, основанных на точных совпадениях, наиболее распространенным названием должности является «Chief Executive Officer» (главный исполнительный директор), за которым следуют другие должности старшего руководящего звена, такие как «President» (президент) и «Founder» (основатель). Следовательно, субъект этой профессиональной сети имеет достаточно широкие контакты в среде предпринимателей и лидеров бизнеса. Наиболее распространенными компонентами в названиях должностей являются «Engineer» (инженер) и «Chief» (главный). Компонент «Chief» (главный) коррелирует с предыдущей мыслью о связях с высшими должностными лицами в компаниях, тогда как компонент «Engineer» (инженер) представляет несколько иное понимание природы профессиональной сети. Конечно, слово «Engineer» не является составной частью наиболее распространенного названия должности, но оно встречается в большом количестве названий других должностей — например, «Senior Software Engineer» (старший инженер-программист) и «Software Engineer» (инженер-программист), — которые занимают верхние строчки списка. То есть субъект этой сети, по-видимому, также имеет широкие связи с техническими специалистами.

Именно такая возможность интерпретации результатов анализа данных о контактах в профессиональной сети мотивирует необходимость создания алгоритма определения сходства или кластеризации. Следующий раздел продолжает исследования в этом направлении.

Нормализация и подсчет местоположений

Хотя LinkedIn содержит информацию о том, как связаться с нашими контактами, у нас больше нет возможности экспортировать географическую информацию. В результате перед нами встает типичная для информатики проблема: что делать при нехватке нужной информации. А если часть географической информации неоднозначна или имеет несколько возможных представлений? Например, все названия — «New York», «New York City», «NYC», «Manhattan» и «New York Metropolitan Area» — относятся к одному и тому же географическому местоположению, но для правильного их учета может потребоваться нормализация.

Задача устранения неоднозначности в географических ссылках не имеет простого общего решения. Нью-Йорк — достаточно большой город, чтобы уверенно предположить, что под названием «New York» подразумевается «New York City». А что вы сказали бы о названии «Smithville»? В Соединенных Штатах есть много городов с таким названием, и даже по нескольку в некоторых штатах, поэтому для правильного определения географического местоположения требуется дополнительный контекст. Конечно, в LinkedIn нечасто можно встретить такое неоднозначное название, как «Greater Smithville Area», но этот пример наглядно иллюстрирует общую проблему устранения неоднозначности географической привязки и ее преобразования в определенный набор координат.

Задача устранения неоднозначности и геокодирования местонахождений контактов в LinkedIn решается немного проще, чем ее обобщенная форма, потому что большинство профессионалов склонны ассоциировать себя с крупными городами, число которых относительно невелико. Даже притом, что это не всегда так, в общем случае можно считать, что местоположение, указанное в профиле LinkedIn, является относительно известным местоположением и, вероятно, будет «самым популярным» большим городом с этим названием.

Можно ли сделать обоснованные предположения в случаях, когда точная информация отсутствует? Есть ли другой способ попытаться понять, где живут и работают ваши контакты, теперь, когда LinkedIn не экспортирует их местоположений?

Как оказывается, обоснованные предположения можно сделать, если выполнить географический поиск по адресу компании, в которой работают ваши контакты. Этот подход неприменим для компаний, которые не указывают свой адрес публично. Также, если работодатель вашего контакта имеет офисы в нескольких городах, географический поиск может вернуть неправильный адрес. Тем не

менее этот способ можно использовать на начальном этапе и с его помощью попробовать узнать географическое расположение ваших контактов.

С этой целью можно установить пакет для Python `geopy`, выполнив команду `pip install geopy`. Он реализует обобщенный механизм передачи географических названий и получения списков координат, которые могут соответствовать им. Пакет `geopy` фактически является промежуточным звеном для доступа к нескольким веб-службам, таким как Bing и Google, которые выполняют геокодирование, и главное его преимущество заключается в том, что он предлагает стандартизированный API доступ к различным службам геокодирования и избавляет от необходимости вручную формировать запросы и обрабатывать ответы. Отличной отправной точкой для знакомства с `geopy` является документация¹, доступная в репозитории GitHub.

Пример 4.6 иллюстрирует, как использовать `geopy` с интерфейсом геокодирования Google Maps. Для его опробования вам потребуется получить ключ API в Google Developers Console².

Пример 4.6. Геокодирование с использованием Google Maps API

```
from geopy import geocoders # pip install geopy

GOOGLEMAPS_APP_KEY = '' # Получите ключ по адресу https://console.developers.
google.com/
g = geocoders.GoogleV3(GOOGLEMAPS_APP_KEY)

location = g.geocode("O'Reilly Media")
print(location)
print('Lat/Lon: {0}, {1}'.format(location.latitude,
                                  location.longitude))
print('https://www.google.ca/maps/@{0},{1},17z'.format(location.latitude,
                                                         location.longitude))
```

Далее, как показано в примере 4.7, выполним обход наших контактов и произведем поиск географических координат по содержимому поля «Companу» в файле CSV. Ниже приводится пример результатов работы этого сценария, иллюстрирующих неоднозначность такого географического названия, как «Nashville»:

```
[(u'Nashville, TN, United States', (36.16783905029297, -86.77816009521484)),
 (u'Nashville, AR, United States', (33.94792938232422, -93.84703826904297)),
```

¹ <http://bit.ly/1a1m7Ka>.

² <http://bit.ly/2EGbF15>.

```
(u'Nashville, GA, United States', (31.206039428710938, -83.25031280517578)),
(u'Nashville, IL, United States', (38.34368133544922, -89.38263702392578)),
(u'Nashville, NC, United States', (35.97433090209961, -77.96495056152344)))
```

Пример 4.7. Геокодирование названий компаний

```
import os
import csv
from geopy import geocoders # pip install geopy

GOOGLEMAPS_APP_KEY = '' # Получите ключ по адресу https://console.developers.
google.com/
g = geocoders.GoogleV3(GOOGLEMAPS_APP_KEY)

# Укажите здесь путь к вашему файлу 'Connections.csv'
CSV_FILE = os.path.join('resources', 'ch03-linkedin', 'Connections.csv')

csvReader = csv.DictReader(open(CSV_FILE), delimiter=',', quotechar='')
contacts = [row for row in csvReader]

for i, c in enumerate(contacts):
    progress = '{0:3d} of {1:3d} - '.format(i+1, len(contacts))
    company = c['Company']
    try:
        location = g.geocode(company, exactly_one=True)
    except:
        print('... Failed to get a location for {0}'.format(company))
        location = None

    if location != None:
        c.update([('Location', location)])
        print(progress + company[:50] + ' -- ' + location.address)
    else:
        c.update([('Location', None)])
        print(progress + company[:50] + ' -- ' + 'Unknown Location')
```

Вот как примерно выглядят результаты работы сценария из примера 4.7:

```
40 of 500 - TE Connectivity Ltd. -- 250 Eddie Jones Way, Oceanside, CA...
41 of 500 - Illinois Tool Works -- 1568 Barclay Blvd, Buffalo Grove, IL...
42 of 500 - Hewlett Packard Enterprise -- 15555 Cutten Rd, Houston, TX...
... Failed to get a location for International Business Machines
43 of 500 - International Business Machines -- Unknown Location
44 of 500 - Deere & Co. -- 1 John Deere Pl, Moline, IL 61265, USA
... Failed to get a location for Affiliated Managers Group Inc
45 of 500 - Affiliated Managers Group Inc -- Unknown Location
46 of 500 - Mettler Toledo -- 1900 Polaris Pkwy, Columbus, OH 43240, USA
```

Далее в этой главе мы будем использовать координаты, возвращаемые службой геокодирования, в алгоритме кластеризации, который может оказаться непо-

хим инструментом для анализа вашей профессиональной сети. Но прежде рассмотрим один интересный способ визуализации географической информации, который называется *картограммой*.

В зависимости от количества обращений к API геокодирования программному коду в примере 4.7 может потребоваться некоторое время на выполнение. Поэтому, чтобы в дальнейшем не тратить это время на повторное получение географических координат, сохраним результаты в удобном и универсальном формате JSON, как показано в примере 4.8.

Пример 4.8. Сохранение географических координат в формате JSON

```
CONNECTIONS_DATA = 'linkedin_connections.json'

# Выполнить обход контактов и изменить информацию о местоположении,
# добавив значения широты и долготы
def serialize_contacts(contacts, output_filename):
    for c in contacts:
        location = c['Location']
        if location != None:
            # Преобразовать местоположение в строку для сериализации
            c.update(['Location', location.address])
            c.update(['Lat', location.latitude])
            c.update(['Lon', location.longitude])

    f = open(output_filename, 'w')
    f.write(json.dumps(contacts, indent=1))
    f.close()
    return

serialize_contacts(contacts, CONNECTIONS_DATA)
```

Эти данные пригодятся нам в разделе «Кластеризация методом k средних».

Визуализация местоположения с помощью картограмм

Картограмма¹ — это картографическое изображение, визуально показывающее величину какого-либо значения в пределах территории на карте. Например, на карте США можно масштабировать размеры отдельных штатов, так чтобы они были больше или меньше, в зависимости, например, от уровня ожирения, уровня бедности, количества миллионеров или любого другого показателя. Получившаяся картограмма не обязательно будет отражать целостное пред-

¹ <http://bit.ly/1a1m5Ss>.

ставление о географии, поскольку отдельные штаты не будут соответствовать друг другу из-за масштабирования. Тем не менее с ее помощью можно получить общее представление об уровне показателя, вызвавшем масштабирование каждого штата.

В специализированной разновидности картограмм, которая называется *картограммой Дорлинга*¹, каждая единица площади на карте замещается геометрической фигурой, например кругом, размещаемой на карте примерно в том же месте и масштабируемой в соответствии со значением показателя. Картограммы Дорлинга также часто называют «географическими пузырьковыми диаграммами». Это отличный инструмент визуализации, основанный на наших интуитивных представлениях о том, где должна находиться информация на поверхности двумерного отображения, и позволяющий кодировать параметры, используя такие понятные свойства фигур, как площадь и цвет.

Служба геокодирования Google Maps включает в результаты название штата для каждого найденного соответствия. Давайте воспользуемся этой информацией и построим картограмму Дорлинга профессиональной сети, в которой размер каждого штата будет соответствовать количеству контактов, проживающих в нем. Современный инструмент визуализации D3² включает почти все необходимое для создания картограмм Дорлинга и предлагает средства расширения визуализации для включения других показателей, если это потребуется. Кроме того, D3 поддерживает несколько других способов визуализации, передающих географическую информацию, таких как тепловые карты, карты символов и хороплет-карты (choropleth maps), которые легко адаптируются к рабочим данным.

На самом деле есть только одна задача, которую необходимо решить, чтобы отобразить количество контактов на карте, — подсчитать их количество, сгруппировав по названиям штатов в ответах службы геокодирования.

Геокодер Google Maps возвращает структурированный результат, из которого без труда можно извлекать названия штатов.

Пример 4.9 показывает, как организовать парсинг ответа геокодера и записать результаты в файл JSON, который потом можно загрузить в механизм визуализации картограммы Дорлинга на основе D3. Поскольку, подготавливая данные для визуализации, мы ориентируемся только на штаты США, нам нужно отфильтровать местоположения из других стран. Для этого была на-

¹ <http://stanford.io/1a1m5SA>.

² <http://bit.ly/1a1kGvo>.

писана вспомогательная функция `checkIfUSA`, которая возвращает `True`, если местоположение находится в США.

Пример 4.9. Парсинг ответа геокодера Google Maps с использованием регулярных выражений

```
def checkIfUSA(loc):
    if loc == None: return False
    for comp in loc.raw['address_components']:
        if 'country' in comp['types']:
            if comp['short_name'] == 'US':
                return True
    else:
        return False

def parseStateFromGoogleMapsLocation(loc):
    try:
        address_components = loc.raw['address_components']
        for comp in address_components:
            if 'administrative_area_level_1' in comp['types']:
                return comp['short_name']
    except:
        return None

results = {}
for c in contacts:
    loc = c['Location']
    if loc == None: continue
    if not checkIfUSA(loc): continue
    state = parseStateFromGoogleMapsLocation(loc)
    if state == None: continue
    results.update({loc.address : state})

print(json.dumps(results, indent=1))
```

Пример результатов, что приводится ниже, иллюстрирует эффективность этого приема:

```
{
  "1 Amgen Center Dr, Thousand Oaks, CA 91320, USA": "CA",
  "1 Energy Plaza, Jackson, MI 49201, USA": "MI",
  "14460 Qorvo Dr, Farmers Branch, TX 75244, USA": "TX",
  "1915 Rexford Rd, Charlotte, NC 28211, USA": "NC",
  "1549 Ringling Blvd, Sarasota, FL 34236, USA": "FL",
  "539 S Main St, Findlay, OH 45840, USA": "OH",
  "1 Ecolab Place, St Paul, MN 55102, USA": "MN",
  "N Eastman Rd, Kingsport, TN 37664, USA": "TN",
  ...
}
```

Получив возможность надежно выделять сокращенные названия штатов из контактов в LinkedIn, теперь можно вычислить частоту появления каждого из них, а это все, что нужно для построения картограммы Дорлинга с помощью D3. Пример такой картограммы для профессиональной сети показан на рис. 4.3. Несмотря на то что картограмма изображает лишь множество кругов на карте, глядя на нее, нетрудно догадаться, какие круги каким штатам соответствуют (обратите внимание, что на многих картограммах штаты Аляска и Гавайи отображаются в нижнем левом углу, так же как на многих картах, где эти штаты отображаются в виде отдельных врезок). При наведении указателя мыши на кружки появляются всплывающие подсказки, по умолчанию отображающие название штата, но их без труда можно изменить, следуя стандартным приемам использования D3. Чтобы подготовить данные для передачи в D3, требуется чуть больше, чем просто подсчитать частоты по штатам и сериализовать их в файл JSON.

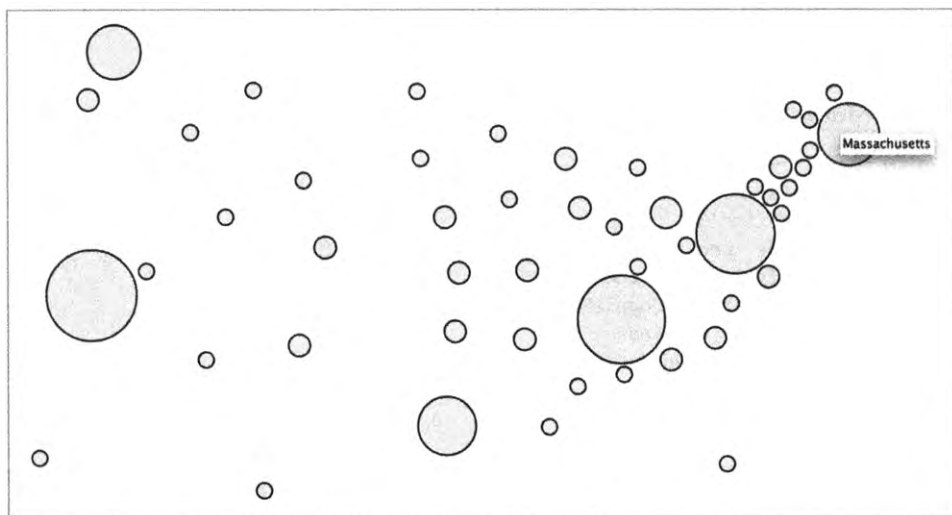


Рис. 4.3. Картограмма Дорлинга местоположений, извлеченных из профессиональной сети LinkedIn, — при наведении указателя мыши на кружки появляются всплывающие подсказки, по умолчанию отображающие название штата (в данном примере указатель мыши наведен на кружок, соответствующий штату Массачусетс)



Часть кода, реализующего создание картограммы Дорлинга на основе контактов из LinkedIn, в этом разделе опущена для краткости, но она включена в исходный код примера Jupyter Notebook для этой главы.

4.3.2. Измерение степени сходства

Познакомившись с некоторыми тонкостями, связанными с нормализацией данных, обратимся теперь к задаче измерения степени сходства, которая является основой кластеризации. Самое важное решение, которое мы должны принять перед кластеризацией набора строк (в данном случае названий должностей), — какую меру сходства использовать. Существует множество мер, позволяющих оценить сходство строк, и выбор наиболее подходящей из них во многом зависит от преследуемых целей.

Все эти меры нетрудно определить и вычислить самостоятельно, тем не менее я воспользуюсь удобной возможностью и представлю вам набор инструментов Natural Language Toolkit (NLTK)¹ для Python, который вы будете рады иметь в своем арсенале средств анализа социальных сетей. По аналогии с другими пакетами для Python установить NLTK можно простой командой `pip install nltk`.



В зависимости от особенностей использования NLTK может потребоваться загрузить дополнительные наборы данных, не входящие в пакет по умолчанию. Если вы не используете виртуальную машину, сопровождающую эту книгу, выполните команду `nltk.download()`, чтобы загрузить все эти дополнительные данные для NLTK. Подробности ищите в документации².

Вот несколько распространенных мер сходства, реализованных в NLTK, которые можно использовать для сравнения названий должностей.

Редакционное расстояние

Редакционное расстояние (edit distance), также известное как расстояние Левенштейна³, — это минимальное количество операций вставок, удалений и замен, необходимое для преобразования одной строки в другую. Например, для преобразования *dad* в *bad* требуется одна операция замены (первой буквы *d* на *b*), в результате редакционное расстояние между этими двумя словами равно 1. NLTK включает реализацию алгоритма редакционного расстояния в виде функции `nltk.metrics.distance.edit_distance`.

Фактическое редакционное расстояние между двумя строками полностью отличается от числа операций, требуемых для *вычисления* редакционного

¹ <http://bit.ly/1a1mc0m>.

² <http://bit.ly/1a1mcgV>.

³ <http://bit.ly/1JtgTWJ>.

расстояния; как правило, для вычисления редакционного расстояния требуется выполнить примерно $M \times N$ операций, где M и N — это длины строк. Иначе говоря, вычисление редакционного расстояния может оказаться дорогостоящей процедурой с точки зрения времени, поэтому при анализе нетривиальных данных используйте его с умом.

Сходство n -грамм

n -граммы предлагают простой способ представления текста в виде последовательности всех возможных групп из n лексем. Этот способ обеспечивает основу для подсчета словосочетаний. Существует много вариантов вычисления сходства с применением n -грамм, но мы остановимся на наиболее простом из них — делении двух строк на последовательности биграмм (2-грамм) и оценке сходства путем подсчета количества общих биграмм в них, как показано в примере 4.10.



Подробное рассмотрение n -грамм и словосочетаний вы найдете в разделе «Анализ биграмм на естественном языке» в главе 5.

Пример 4.10. Использование NLTK для вычисления биграмм

```
from nltk.util import bigrams

ceo_bigrams = list(bigrams("Chief Executive Officer".split(),
                           pad_left=True, pad_right=True))
cto_bigrams = list(bigrams("Chief Technology Officer".split(),
                           pad_left=True, pad_right=True))

print(ceo_bigrams)
print(cto_bigrams)

print(len(set(ceo_bigrams).intersection(set(cto_bigrams))))
```

Следующие результаты иллюстрируют, как выделяются биграммы и как определяется пересечение множеств биграмм для двух разных названий должностей:

```
[(None, 'Chief'), ('Chief', 'Executive'), ('Executive', 'Officer'),
 ('Officer', None)]
[(None, 'Chief'), ('Chief', 'Technology'), ('Technology', 'Officer'),
 ('Officer', None)]
2
```

Именованные аргументы `pad_right` и `pad_left` позволяют организовать сопоставление начальных и конечных слов в названиях по отдельности. Благодаря им в наборы будут включены такие биграммы, как (`None`, `'Chief'`), дающие важные совпадения в названиях должностей. В NLTK имеется довольно полный набор функций оценки биграмм и триграмм (3-грамм), организованных в классы `BigramAssociationMeasures` и `TrigramAssociationMeasures`, в модуле `nlk.metrics.association`.

Расстояние Жаккара

Часто сходство можно вычислить как сходство двух множеств, где под *множеством* понимается неупорядоченная коллекция элементов. Метрика сходства Жаккара (Jaccard distance) выражает сходство двух множеств и определяется как отношение пересечения множеств к их объединению. Математически сходство Жаккара записывается как:

$$\frac{|\text{Множество 1} \cap \text{Множество 2}|}{|\text{Множество 1} \cup \text{Множество 2}|},$$

где числитель — это число общих элементов в двух множествах (мощность их пересечения), а знаменатель — общее количество разных элементов в двух множествах (мощность их объединения). Проще говоря, это число уникальных элементов, общих для обоих множеств, деленное на общее число уникальных элементов. Сходство Жаккара является разумным способом получения нормализованной оценки сходства. Для оценки сходства двух строк часто используется прием вычисления сходства Жаккара между множествами n -грамм, в том числе и униграмм (1-грамм).

Учитывая, что метрика сходства Жаккара измеряет близость двух множеств, их несходство можно измерить, вычитая это значение из 1,0, и получить оценку, известную как расстояние Жаккара.



Кроме этих удобных мер сходства и наряду с другими многочисленными утилитами, пакет NLTK предлагает класс `nlk.FreqDist`. Он вычисляет распределение частот, подобно `collections.Counter` из стандартной библиотеки Python.

Вычисление сходства является важнейшим аспектом любого алгоритма кластеризации, и вы легко сможете опробовать разные эвристики сходства в рамках своей работы, как только получите более полное представление об анализи-

руемых данных. В следующем разделе описывается сценарий кластеризации названий должностей с использованием метрики сходства Жаккара.

4.3.3. Алгоритмы кластеризации

Теперь, имея все необходимое для нормализации данных и вычисления сходства, получим исходные данные из LinkedIn и определим некоторые кластеры, чтобы получить дополнительные сведения о динамике профессиональной сети. Если вы захотите понять, помогла ли вам ваша профессиональная сеть встретить «нужных людей»; оценить, насколько ваши контакты вписываются в социально-экономическую систему с определенной профессиональной направленностью; или определить, есть ли лучшее место, куда можно переехать или где можно открыть удаленный офис для расширения бизнеса, вы обязательно найдете ценную информацию в данных, извлеченных из профессиональной сети. В оставшейся части этого раздела мы рассмотрим несколько подходов к кластеризации, продолжив изучение задачи группировки схожих названий должностей.

Жадная кластеризация

Теперь, понимая важность пересечений в названиях должностей, давайте попробуем сгруппировать их, сравнивая их с использованием меры расстояния Жаккара в продолжение примера 4.5. Пример 4.11 группирует ваши контакты по сходству названий должностей, а затем отображает их. Сначала рассмотрите код в примере 4.11, обратив особое внимание на вложенный цикл, вызывающий функцию `DISTANCE`, а затем переходите к его обсуждению.

Пример 4.11. Кластеризация названий должностей с использованием жадной эвристики

```
import os
import csv
from nltk.metrics.distance import jaccard_distance

# Укажите здесь путь к вашему файлу 'Connections.csv'
CSV_FILE = os.path.join('resources', 'ch03-linkedin', 'Connections.csv')

# Попробуйте настроить пороговое значение DISTANCE_THRESHOLD
# и использовать разные алгоритмы вычисления расстояния
DISTANCE_THRESHOLD = 0.6
DISTANCE = jaccard_distance
```

```
def cluster_contacts_by_title():

    transforms = [
        ('Sr.', 'Senior'),
        ('Sr', 'Senior'),
        ('Jr.', 'Junior'),
        ('Jr', 'Junior'),
        ('CEO', 'Chief Executive Officer'),
        ('COO', 'Chief Operating Officer'),
        ('CTO', 'Chief Technology Officer'),
        ('CFO', 'Chief Finance Officer'),
        ('VP', 'Vice President'),
    ]

    separators = ['/', ' and ', ' & ', '|', ',']

    # Нормализовать и/или заменить известные сокращения
    # и сконструировать список типичных названий.
    all_titles = []
    for i, _ in enumerate(contacts):
        if contacts[i]['Position'] == '':
            contacts[i]['Position'] = ''
            continue
        titles = [contacts[i]['Position']]
        # развернуть список
        titles = [item for sublist in titles for item in sublist]
        for separator in separators:
            for title in titles:
                if title.find(separator) >= 0:
                    titles.remove(title)
                    titles.extend([title.strip() for title in
                                   title.split(separator) if title.strip() != ''])

        for transform in transforms:
            titles = [title.replace(*transform) for title in titles]

        contacts[i]['Position'] = titles
        all_titles.extend(titles)

    all_titles = list(set(all_titles))

    clusters = {}
    for title1 in all_titles:
        clusters[title1] = []
        for title2 in all_titles:
            if title2 in clusters[title1] or title2 in clusters and title1
               in clusters[title2]:
                continue
            distance = DISTANCE(set(title1.split()), set(title2.split()))
```



```

        if distance < DISTANCE_THRESHOLD:
            clusters[title1].append(title2)

# Развернуть кластеры

clusters = [clusters[title] for title in clusters if len(clusters[title]) > 1]

# Собрать контакты в этих кластерах и сгруппировать их

clustered_contacts = {}
for cluster in clusters:
    clustered_contacts[tuple(cluster)] = []
    for contact in contacts:
        for title in contact['Position']:
            if title in cluster:
                clustered_contacts[tuple(cluster)].append('{0} {1}'.format(
                    contact['FirstName'], contact['LastName'][0]))

return clustered_contacts

clustered_contacts = cluster_contacts_by_title()

for titles in clustered_contacts:
    common_titles_heading = 'Common Titles: ' + ', '.join(titles)

    descriptive_terms = set(titles[0].split())
    for title in titles:
        descriptive_terms.intersection_update(set(title.split()))
    if len(descriptive_terms) == 0: descriptive_terms = ['***No words
                                                         in common***']

    descriptive_terms_heading = 'Descriptive Terms: '
        + ', '.join(descriptive_terms)
    print(common_titles_heading)
    print('\n'+descriptive_terms_heading)
    print('-' * 70)
    print('\n'.join(clustered_contacts[titles]))
    print()

```

Код начинается с выделения комбинированных названий и их нормализации с использованием списка типичных сокращений. Затем вложенный цикл перебирает названия и группирует их в соответствии с пороговым значением метрики сходства Жаккара, которая определяется присваиванием переменной `DISTANCE` ссылки на функцию `jaccard_distance` — такое решение облегчает подстановку другого алгоритма вычисления расстояния для экспериментов. Этот короткий цикл выполняет большую часть работы: он сравнивает все названия друг с другом.

Если расстояние между названиями, определяемое выбранным алгоритмом, оказывается «достаточно маленьким», мы *жадно* объединяем их. В данном случае под словом «жадно» подразумевается, что элемент добавляется в первый подходящий кластер и не предпринимается попыток проверить наличие другого кластера с лучшим соответствием и пересмотреть степень соответствия при появлении такого кластера позже. Несмотря на сугубую прагматичность подхода, он дает очень хорошие результаты. Очевидно, что для его успеха решающее значение имеет выбор эффективной эвристики сходства, но, учитывая характер вложенного цикла, чем меньшее количество раз будет вызвана функция оценки, тем быстрее выполнится код (основная проблема для нетривиальных наборов данных). К этой проблеме мы еще вернемся в следующем разделе, но обратите внимание, что с целью избежать повторения ненужных вычислений здесь мы используем некоторую условную логику.

Остальной код в листинге просто отыскивает контакты с определенными названиями должностей и группирует их для отображения, но в кластеризации данных есть еще один нюанс: *часто бывает нужно присвоить каждому кластеру осмысленную метку*. Рабочая реализация вычисляет метки, отыскивая пересечение слов в названиях должностей в каждом кластере, что кажется разумным, учитывая, что это наиболее очевидная общая основа. В других случаях вам наверняка придется использовать какой-то другой подход.

Результаты, возвращаемые этим кодом, объединяют людей, которые, вероятно, выполняют общие обязанности, если судить по названиям их должностей. Как уже отмечалось, эта информация может пригодиться, если вы планируете мероприятие, включающее «панель генерального директора», пытаетесь выяснить, кто сможет помочь вам сделать следующий карьерный шаг, или хотите определить, достаточно ли тесно вы связаны с другими профессионалами, учитывая ваши собственные должностные обязанности и будущие устремления. Ниже приводятся сокращенные результаты для примера профессиональной сети:

Common Titles: Sociology Professor, Professor
Descriptive Terms: Professor

Kurtis R.
Patrick R.
Gerald D.
April P.
...

Common Titles: Petroleum Engineer, Engineer
Descriptive Terms: Engineer

Timothy M.
Eileen V.
Lauren G.
Erin C.
Julianne M.
...

Анализ времени выполнения



В этом разделе довольно подробно обсуждаются вычислительные детали кластеризации, и его не следует рассматривать как нечто обязательное для чтения, потому что он может понравиться не всем. Читатели эту главу в первый раз могут смело пропустить этот раздел и прочитать его позже.

В худшем случае вложенный цикл в примере 4.11, вычисляющий расстояния, будет вызван $\text{len}(\text{all_titles}) * \text{len}(\text{all_titles})$ раз — иными словами, наш алгоритм обладает квадратичной временной сложностью $O(n^2)$. Подход с вложенным циклом, попарно сравнивающим все имеющиеся элементы, плохо масштабируется на случаи с большим значением n , но, учитывая, что уникальное число названий должностей в вашей профессиональной сети вряд ли будет очень большим, он не должен сильно сказываться на производительности. Это обстоятельство может показаться не важным — в конце концов, это просто вложенный цикл, но проблема в том, что количество сравнений, выполняемых алгоритмом со сложностью $O(n^2)$ при обработке входного набора, увеличивается в экспоненциальной прогрессии пропорционально количеству элементов в множестве. Например, для небольшого входного набора из 100 названий должностей потребуется выполнить только 10 000 операций оценки, но для 10 000 названий потребуется выполнить уже 100 000 000 таких операций. В конечном итоге объем вычислений может стать слишком большим даже для мощного оборудования.

В таком затруднительном положении, когда алгоритм плохо масштабируется, первым желанием обычно бывает максимально уменьшить значение n . Но не всегда удастся уменьшить его настолько, чтобы сделать решение масштабируемым, потому что внутри все еще присутствует алгоритм $O(n^2)$. В действительности, чтобы справиться с проблемой, нужно придумать новый алгоритм, имеющий сложность $O(k \times n)$, где k намного меньше n и представляет накладные расходы, которые растут гораздо медленнее, чем n . Как и в любой другой инженерной задаче, между производительностью и качеством существует определенный баланс, *найти который порой довольно сложно*. На самом деле, многие компании, занимающиеся анализом данных и успешно внедрившие

масштабируемые алгоритмы сопоставления записей с высокой точностью, считают, что их конкретные подходы являются конфиденциальной информацией (коммерческой тайной), поскольку они дают им определенные конкурентные преимущества.

В ситуациях, когда алгоритм $O(n^2)$ просто неприемлем, можно попробовать переписать вложенные циклы так, чтобы для оценки выбирались случайные экземпляры, и таким способом уменьшить число сравнений до $O(k \times n)$, где k — размер выборки. Однако по мере приближения размера выборки к n сложность выполнения будет приближаться к $O(n^2)$. Следующие изменения в примере 4.11 демонстрируют, как выглядит реализация метода случайной выборки в коде; наиболее важные изменения выделены жирным. Основной вывод: в каждой итерации внешнего цикла выполняется ограниченное количество итераций внутреннего цикла:

```
# ...обрезано...

all_titles = list(set(all_titles))
clusters = {}
for title1 in all_titles:
    clusters[title1] = []
    for sample in range(SAMPLE_SIZE):
        title2 = all_titles[random.randint(0, len(all_titles)-1)]
        if title2 in clusters[title1] or clusters.has_key(title2) and title1
            in clusters[title2]:
            continue
        distance = DISTANCE(set(title1.split()), set(title2.split()))
        if distance < DISTANCE_THRESHOLD:
            clusters[title1].append(title2)

# ...обрезано...
```

Другое решение — случайно разбить данные на n сегментов (где n — некоторое число, обычно меньшее или равное квадратному корню числа элементов в множестве), выполнить кластеризацию в каждом из этих сегментов, а затем, при желании, объединить результаты. Например, для множества с 1 миллионом элементов алгоритму со сложностью $O(n^2)$ потребуется выполнить триллион логических операций, но, если разбить 1 миллион элементов на 1000 сегментов по 1000 элементов в каждом, тогда для кластеризации всех отдельных сегментов потребуется только миллиард операций. (То есть 1000×1000 сравнений в каждом сегменте для всех 1000 сегментов.) Миллиард — это все еще большое число, но оно на три порядка меньше триллиона, и это существенное улучшение (хотя в некоторых ситуациях этого может быть недостаточно).

Кроме выборки и распределения по сегментам есть много других подходов, которые иногда лучше справляются с задачей уменьшения размерности данных. Например, в идеальном случае можно сравнивать все элементы в множестве, а чтобы избежать сложности $O(n^2)$ при больших значениях n — выбрать конкретный метод, в зависимости от реальных ограничений и идей, которые вы, вероятно, получите в результате экспериментов и изучения конкретной области. Рассматривая возможные варианты, имейте в виду, что область машинного обучения предлагает множество методов решения именно таких проблем масштабирования с использованием различных видов вероятностных моделей и сложных методов выборки. В разделе «Кластеризация методом k средних» вы познакомитесь с довольно простым и известным алгоритмом кластеризации методом k средних, реализующим универсальный метод обучения без учителя для кластеризации многомерного пространства. Позже мы используем этот метод для группировки контактов по географическому местоположению.

Иерархическая кластеризация

В примере 4.11 был представлен простой подход к кластеризации, главным образом как упражнение для изучения основных аспектов задачи. Теперь, когда вы познакомились с основными принципами, пришло время представить вам еще два распространенных алгоритма кластеризации, с которыми вы регулярно будете сталкиваться в своей карьере исследователя данных и применять их в различных ситуациях: иерархическая кластеризация и кластеризация методом k средних.

Иерархическая кластеризация внешне напоминает жадную эвристику, уже использовавшуюся нами, тогда как метод k средних радикально отличается от нее. В оставшейся части этой главы основное внимание будет уделяться методу k средних, но мы рассмотрим теоретические основы обоих подходов, так как вы, скорее всего, столкнетесь с обоими в других книгах. Превосходная реализация обоих подходов доступна в модуле `cluster`, который можно установить командой `pip install cluster`.

Иерархическая кластеризация — это детерминированный метод, который предусматривает вычисление полной матрицы¹ расстояний между всеми элементами, а затем выполняет обход ее элементов и производит кластеризацию в соответствии с минимальным порогом расстояния. *Иерархическим* этот подход называется потому, что в процессе обхода матрицы и кластеризации

¹ Вычисление полной матрицы имеет полиномиальную сложность. Для агломеративной кластеризации сложность часто имеет порядок $O(n^3)$.

элементов создается древовидная структура, которая выражает относительные расстояния между элементами. В литературе этот метод часто называют *агломеративным*, потому что он создает дерево, упорядочивая отдельные элементы данных в кластеры, которые иерархически сливаются с другими кластерами, пока весь набор данных не будет кластеризован в верхней части дерева. Листья дерева представляют кластеризуемые элементы данных, а промежуточные узлы иерархически объединяют эти элементы в кластеры.

Чтобы лучше понять идею агломерации, взгляните на рис. 4.4 ниже и обратите внимание, что люди, такие как «Andrew O.» и «Matthias B.», являются листьями дерева кластеризации, тогда как узлы, такие как «Chief, Technology, Officer» (главный технический директор) объединяют (агломерируют) эти листья в кластеры. Хотя дерево на рис. 4.4 имеет только два уровня, нетрудно представить дополнительный уровень агломерации, представляющий руководителя с названием «Chief, Officer» (главный директор) и объединяющий узлы «Chief, Technology, Officer» (главный технический директор) и «Chief, Executive, Officer» (главный исполнительный директор).

Агломерация — это метод, похожий на метод в примере 4.11, который вместо последовательного построения иерархии использует жадную эвристику, но имеющий принципиальные отличия. Для выполнения иерархической кластеризации может потребоваться значительно больше времени, поэтому вам придется соответствующим образом настроить функцию оценки и определить порог расстояния.¹ Часто агломеративная кластеризация не подходит для больших наборов данных из-за ее непрактично низкой производительности.

Если переписать пример 4.11 с использованием пакета `cluster`, тогда вложенный цикл, выполняющий вычисление `DISTANCE`, можно было бы заменить следующим кодом:

```
# ...обрезано...

# Определение функции оценки
def score(title1, title2):
```

¹ Использование динамического программирования (<http://bit.ly/1a1maFO>) и других оптимизированных методов обработки данных может существенно сократить время выполнения, и одно из преимуществ использования готовых инструментов состоит в том, что они уже реализуют такие оптимизации. Например, учитывая, что расстояния между двумя элементами, такими как названия должностей, почти наверняка будут симметричны, достаточно вычислить только половину матрицы расстояний. Поэтому, даже притом, что сложность алгоритма в целом остается равной $O(n^2)$, вместо n^2 выполняется только $n^2/2$ единиц работы.

```
return DISTANCE(set(title1.split()), set(title2.split()))

# Передать класс с данными и функцию оценки
hc = HierarchicalClustering(all_titles, score)

# Выполнить кластеризацию с учетом заданного порогового расстояния
clusters = hc.getlevel(DISTANCE_THRESHOLD)

# Удалить одиночные кластеры
clusters = [c for c in clusters if len(c) > 1]

# ...обрезано...
```

Если вас интересуют варианты иерархической кластеризации, обязательно познакомьтесь с методом `setLinkageMethod` класса `HierarchicalClustering`, который дает возможность выбирать способ вычисления расстояния между кластерами. Например, с его помощью можно указать, что расстояние между кластерами должно определяться как кратчайшее, наибольшее или среднее расстояние между любыми двумя их компонентами. В зависимости от распределения данных выбор разных способов определения расстояния может приводить к разным результатам.

На рис. 4.4 и 4.5 показаны изображения профессиональной сети в виде дендрограммы и дерева связей узлов соответственно, полученные с помощью D3¹ — современного инструмента визуализации, представленного выше. Схема в виде дерева связей узлов более компактна и, вероятно, лучше подходит для этого конкретного набора данных, тогда как дендрограмма² была бы отличным выбором для тех, кому нужно видеть связи между уровнями в дереве (соответствующими уровням агломерации в иерархической кластеризации) в более сложных наборах данных. Если бы иерархия была более глубокой, дендрограмма выглядела бы предпочтительнее, но в данном случае имеется лишь несколько уровней, поэтому конкретные преимущества одной схемы по сравнению с другой носят в основном эстетический характер. Глядя на эти рисунки, можно только удивляться, какой объем информации становится очевидным, когда есть возможность увидеть простое изображение профессиональной сети.



Код создания дерева связанных узлов и дендрограммы с помощью D3 опущен для краткости, но он включен в исходный код примеров Jupyter Notebook для этой главы.

¹ <http://bit.ly/1a1kGvo>.

² <http://bit.ly/1a1md4B>.

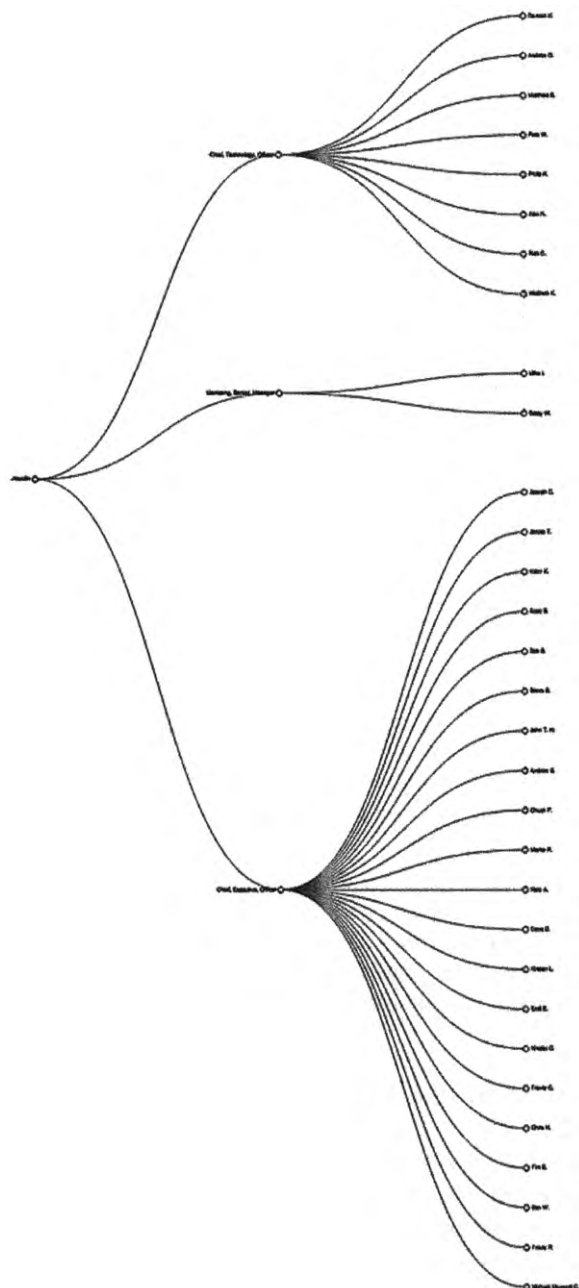


Рис. 4.4. Дендрограмма контактов, сгруппированных по названиям должностей, — обычно дендрограммы явно отражают иерархии

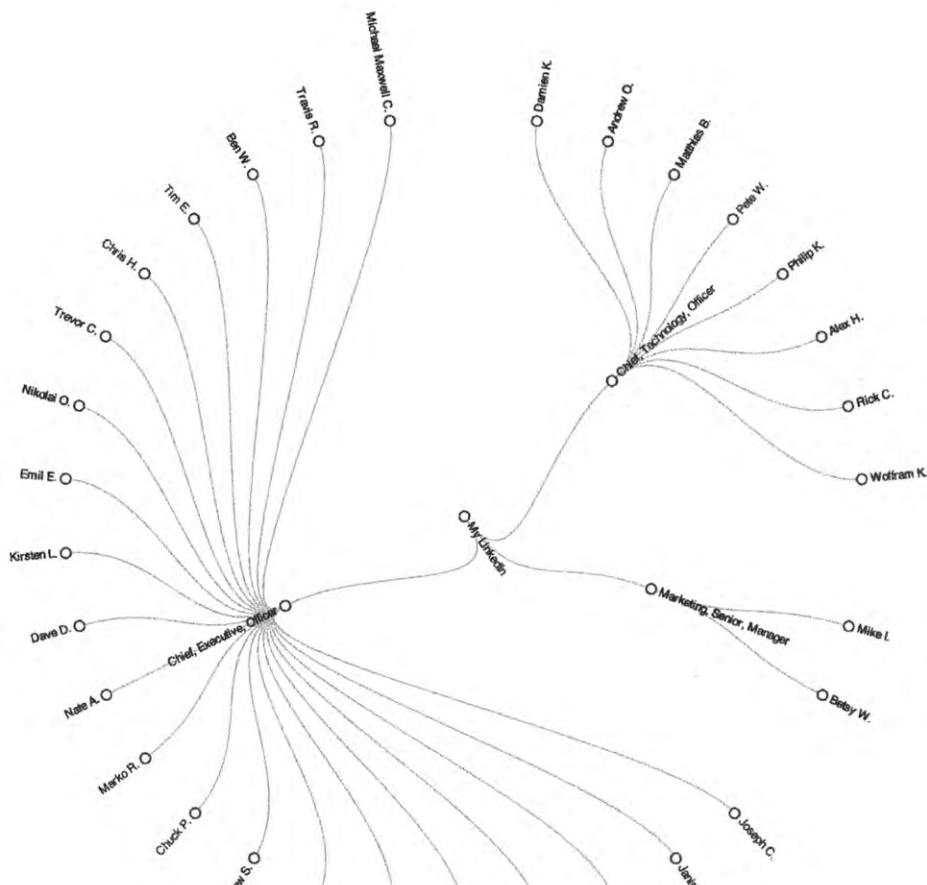


Рис. 4.5. Дерево связей узлов контактов, сгруппированных по названиям должностей, которое передает ту же информацию, что и дендрограмма на рис. 4.4, — обычно деревья связей обеспечивают более компактное и эстетически приятное представление в сравнении с дендрограммами

Кластеризация методом k средних

В отличие от иерархической кластеризации, которая является детерминированным методом, исчерпывающим все возможности и часто дорогостоящим, с вычислительной сложностью порядка $O(n^3)$, кластеризация методом k средних имеет меньшую сложность, около $O(k \times n)$. Даже при больших значениях k экономия получается более чем существенной. Экономия достигается за счет вычисления приблизительных результатов, которые все еще достаточно близки к истине. Идея заключается в том, чтобы сгруппировать многомерное

пространство, содержащее n точек, в k кластеров, выполнив следующую последовательность шагов:

1. Выбрать k случайных точек в пространстве данных, которые будут использоваться для формирования k кластеров: $K_1, K_2, \dots, K_k$.
2. Связать каждую из n точек с ближайшим кластером K_n , фактически выполнив $k \times n$ сравнений.
3. Для каждого из k кластеров вычислить *центроид*¹ — среднее значение кластера — и присвоить это значение кластеру K_i . (То есть в каждой итерации требуется вычислить « k средних».)
4. Повторять шаги 2–3, пока принадлежность к кластерам продолжает изменяться. Часто сходимость наступает после относительно небольшого числа итераций.

Кому-то из вас может быть трудно сразу понять суть алгоритма k средних, поэтому на рис. 4.6 приводится диаграмма из онлайн-руководства «Tutorial on Clustering Algorithms»², отображающая каждый его шаг, которая создается интерактивным Java-апплетом. В данном случае за основу взята выборка из 100 точек данных и для параметра k выбрано значение 3, то есть алгоритм создаст три кластера. Обратите внимание, как меняются местоположения квадратов и какие точки включаются в каждый из трех кластеров по мере продвижения алгоритма. Всего алгоритм выполняет девять итераций.

Алгоритм k средних можно применить и к двумерному, и к 2000-мерному пространству, но на практике размерность пространства данных обычно не превышает десяти и чаще всего равна двум или трем. Для пространств с небольшим числом измерений алгоритм k средних может оказаться весьма эффективным методом кластеризации, потому что действует довольно быстро и способен давать достаточно точные результаты. Правда, при этом нужно выбрать подходящее значение для k , что не всегда просто.

В оставшейся части этого раздела мы посмотрим, как кластеризовать и визуализировать профессиональную сеть по географическим координатам, применяя метод k средних и используя для отображения результатов Google Maps³ или Google Earth⁴.

¹ <http://bit.ly/1a1mbcW>.

² <http://bit.ly/1a1mbtp>.

³ <http://bit.ly/1a1mdRV>.

⁴ <http://bit.ly/1a1meFC>.

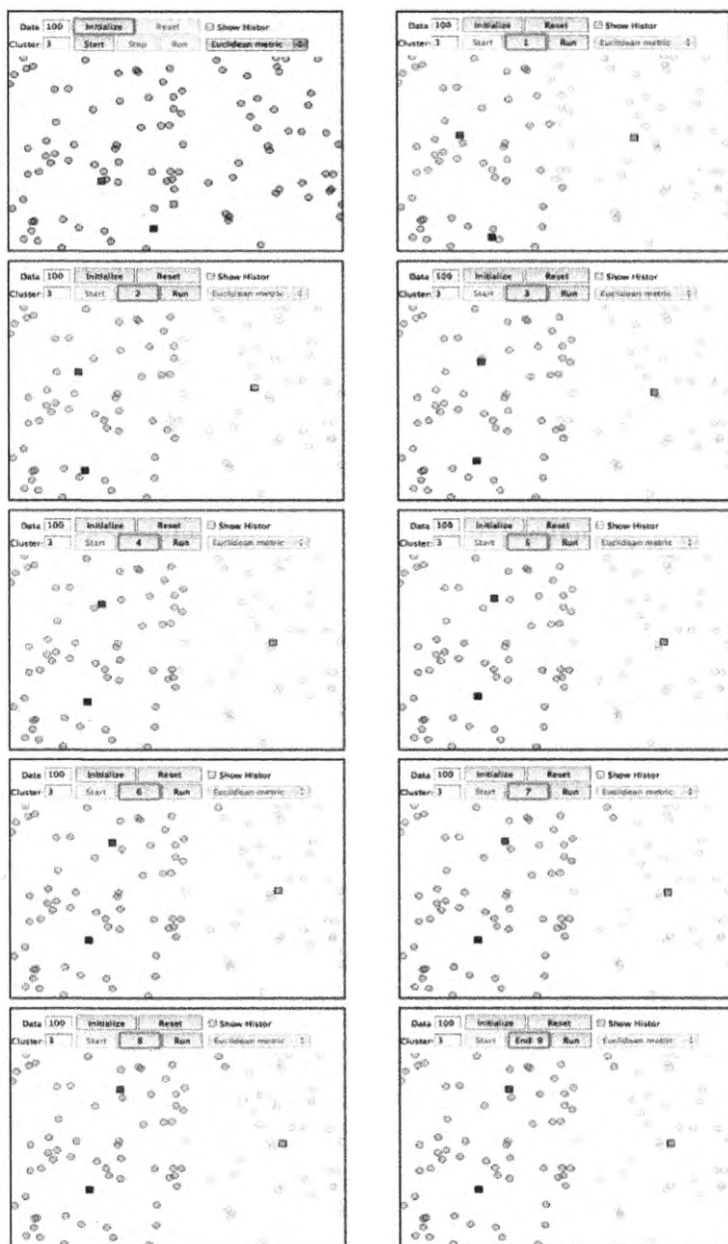


Рис. 4.6. Итерации алгоритма кластеризации методом k средних для 100 точек и $k=3$ — обратите внимание, насколько быстро формируются кластеры в первых нескольких итерациях алгоритма, и в последних итерациях происходит лишь перемещение точек, лежащих вблизи границ кластеров

Визуализация географических кластеров с Google Earth

Полезным упражнением, чтобы увидеть метод k средних в действии, может стать использование его для кластеризации и визуализации профессиональной сети LinkedIn в двумерном пространстве. Благодаря визуализации можно получить общее представление о распределении ваших контактов и имеющихся закономерностях или аномалиях, а также проанализировать кластеры, образованные вашими контактами, работодателями ваших контактов или различными географическими районами, где проживают ваши контакты. Все три подхода могут дать результаты, полезные для различных целей.

Как уже говорилось, из LinkedIn API можно получить информацию о местоположении, описывающую крупные города, такие как «Greater Nashville Area», затем преобразовать ее в географические координаты и вывести в соответствующем формате (например, KML¹), чтобы потом отобразить их на карте с помощью такого инструмента, как Google Earth, поддерживающего интерактивный режим.



Новый механизм Google Maps Engine тоже предоставляет возможность выгрузки данных для целей визуализации.

В число основных действий, которые необходимо выполнить для преобразования контактов LinkedIn в такой формат, как KML, входят: парсинг географического местоположения из профилей контактов и создание KML для визуализации, например, с использованием Google Earth. Пример 4.7 демонстрирует, как преобразовать информацию из профиля в географические координаты, и предоставляет рабочую основу для сбора необходимых нам данных. Кластеризацию можно выполнить с помощью класса `KMeansClustering` из пакета `cluster`, поэтому нам остается только преобразовать данные и результаты кластеризации в формат KML, что реализуется относительно просто с инструментами XML.

Как и в примере 4.11, основная работа, связанная с подготовкой результатов для визуализации, является типовой обработкой данных. Самое интересное спрятано в вызове метода `getclusters` класса `KMeansClustering`. Данное решение группирует контакты по местоположению, а затем использует результаты алгоритма кластеризации для вычисления центроидов. На рис. 4.7 и 4.8 показаны результаты выполнения кода в примере 4.12. Пример начинается

¹ <http://bit.ly/1a1meWb>.

с чтения информации, полученной и сохраненной в формате JSON на этапе геокодирования в примере 4.8.



Рис. 4.7. Карта местоположений всех контактов



Рис. 4.8. Карта местоположений центроидов, полученных в результате кластеризации методом k средних

Пример 4.12. Кластеризация контактов из профессиональной сети LinkedIn на основе информации о местоположении и вывод данных в формате KML для визуализации в Google Earth

```
import simplekml # pip install simplekml
from cluster import KMeansClustering
from cluster.util import centroid

# Данные будут загружаться из прежде сохраненного файла
CONNECTIONS_DATA = 'linkedin_connections.json'

# Извлечь сохраненную информацию о контактах, дополненную географическими
# координатами, из файла или получить ее заново из LinkedIn, если хотите
connections = json.loads(open(CONNECTIONS_DATA).read())

# Объект KML для сохранения контактов
kml_all = simplekml.Kml()

for c in connections:
    location = c['Location']
    if location is not None:
        lat, lon = c['Lat'], c['Lon']
        kml_all.newpoint(name='{} {}'.format(c['FirstName'], c['LastName']),
                          coords=[(lon,lat)]) # coords reversed

kml_all.save('resources/ch03-linkedin/viz/connections.kml')

# Выполнить кластеризацию методом k средних в K кластеров

K = 10

cl = KMeansClustering([(c['Lat'], c['Lon']) for c in connections
                        if c['Location'] is not None])

# Извлечь центроиды из всех K кластеров
centroids = [centroid(c) for c in cl.getclusters(K)]

# Объект KML для хранения местоположений кластеров
kml_clusters = simplekml.Kml()

for i, c in enumerate(centroids):
    kml_clusters.newpoint(name='Cluster {}'.format(i),
                          coords=[(c[1],c[0])]) # coords reversed

kml_clusters.save('resources/ch03-linkedin/viz/kmeans_centroids.kml')
```

Код в примере 4.12 использует библиотеку `simplekml` для Python, чтобы упростить создание объектов KML, и создает на диске два файла в формате KML, которые можно выгрузить в геопространственное приложение, такое как Google Earth. Первый из этих файлов содержит приблизительные местоположения всех ваших контактов в LinkedIn, для которых службе геокодирования удалось определить географические координаты по названиям указанных местоположений.

Затем, после кластеризации методом k средних, в файл KML записываются местоположения 10 центроидов. Вы можете сравнить эти два файла в Google Earth и увидеть, где находятся центроиды кластеров относительно отдельных контактов. Здесь можно обнаружить, что центроиды располагаются в крупных городах. Попробуйте поэкспериментировать с разными значениями k и посмотрите, какое из них лучше подходит для обобщения географического распределения ваших контактов в LinkedIn.

Даже простая визуализация вашей сети может натолкнуть на ранее незаметные идеи, а вычисление географических центроидов может открыть перед вами некоторые интригующие возможности. Например, таким способом можно определить оптимальное место для проведения серии региональных семинаров или конференций. Или, если вы занимаетесь консалтинговым бизнесом и имеете напряженный график поездок, вы сможете выбрать хорошие места для аренды жилья вдали от дома. Аналогично можно отобразить на карте контакты по должностным обязанностям или социально-экономическим показателям, в которые они укладываются согласно их должности и опыту. Кроме многочисленных вариантов визуализации данных о местоположении контактов из вашей профессиональной сети, географическая кластеризация предлагает множество других возможностей, таких как управление цепочками поставок и решение задачи коммивояжера¹, когда требуется минимизировать расходы, связанные с поездками или перемещением товаров из точки в точку.

4.4. Заключительные замечания

В этой главе мы обсудили некоторые важные вопросы, познакомились с фундаментальной идеей кластеризации и рассмотрели разные способы ее применения к данным из профессиональной сети в LinkedIn. Вне всяких сомнений, эта глава была более продвинутой, чем предыдущие, — здесь мы познакомились с такими

¹ <http://bit.ly/1a1mhkF>.

общими проблемами, как нормализация данных, вычисление сходства в нормализованных данных и вычислительная эффективность подходов, используемых в анализе данных. Эта глава действительно была достаточно сложной, чтобы ее можно было усвоить с первого раза, поэтому не расстраивайтесь, если почувствовали себя немного перегруженными. Возможно, придется прочитать ее несколько раз, чтобы полностью усвоить детали, представленные здесь.

Также не забывайте, что для практического применения кластерного анализа не требуется полностью понимать базовую теорию, хотя в целом вы должны стараться понять базовые принципы, лежащие в основе методов, которые используете при анализе социальной сети. Как и в других главах, мы затронули лишь верхушку айсберга; есть много других интересных видов анализа, вообще не требующих кластеризации, которые можно применить к данным из LinkedIn, но не представленных в этой главе, включая простой частотный анализ. И тем не менее вы получили в свое распоряжение отличный мощный инструмент.



Исходный код примеров для этой и всех других глав доступен на GitHub¹ в удобном формате Jupyter Notebook, который вы можете опробовать, не покидая веб-браузера.

4.5. Упражнения

- Выделите немного времени, чтобы получше изучить имеющуюся у вас расширенную информацию из профилей. Было бы интересно попробовать соотнести место работы и обучения и/или проанализировать склонность людей к переездам в определенные районы и из них.
- Попробуйте использовать альтернативные средства из D3, такие как хороплет-карты², для визуализации вашей профессиональной сети.
- Почитайте о новой и интересной спецификации geoJSON³ и о том, как с ее помощью можно создавать интерактивные визуализации на GitHub, генерируя данные geoJSON. Попробуйте применить этот метод к вашей профессиональной сети вместо Google Earth.

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² <http://bit.ly/1a1mg0a>.

³ <http://bit.ly/1a1mggF>.

- Познакомьтесь с инструментом `geodict`¹ и некоторыми другими утилитами в наборе Data Science Toolkit². Сможете ли вы определить географические координаты по произвольному текстовому описанию и отобразить их на карте, чтобы получить представление о происходящем, без необходимости читать этот текст?
- Проанализируйте профили в Twitter или Facebook на наличие географической информации и визуализируйте полученные результаты. Твиты и посты в Facebook часто содержат геометки в структурированных метаданных.
- LinkedIn API дает возможность получить ссылку на аккаунт контакта в Twitter. Сколько ваших контактов в LinkedIn имеют аккаунты Twitter, связанные с их профессиональными профилями? Насколько они активны? Насколько *профессиональны* их онлайн-личности в Twitter с точки зрения потенциального работодателя?
- Попробуйте применить методы кластеризации из этой главы к твитам. Сможете ли вы извлечь значимые сущности из твитов, определить значимую метрику сходства и сгруппировать твиты значимым образом?
- Попробуйте применить методы кластеризации из этой главы к данным из Facebook, таким как лайки и сообщения. Сможете ли вы определить значимую метрику сходства для коллекции лайков вашего друга и сгруппировать лайки значимым образом? Сможете ли вы сгруппировать лайки всех ваших друзей (или самих друзей) значимым образом?

4.6. Онлайн-ресурсы

Ниже приводится список ссылок, упоминавшихся в этой главе, которые могут оказаться полезными для вас:

- Портал Bing Maps³.
- Центроид⁴.
- Галерея примеров для D3.js⁵.

¹ <http://bit.ly/1a1mgxd>.

² <http://bit.ly/1a1mgNK>.

³ <http://bit.ly/1a1m5lq>.

⁴ <http://bit.ly/1a1mbcW>.

⁵ <http://bit.ly/1a1lMal>.

- Data Science Toolkit¹.
- Дендрограммы².
- Пакет геору в репозитории GitHub³.
- Описание Google Maps API⁴.
- Язык разметки Keyhole Markup Language (KML)⁵.
- Расстояние Левенштейна⁶.
- Синтаксис селекторов полей в LinkedIn API⁷.
- Экспортирование данных в LinkedIn⁸.
- Документация с описанием LinkedIn REST API⁹.
- Отображение файлов geoJSON в GitHub¹⁰.
- Страница пакета `python3-linkedin` в каталоге PyPi¹¹.
- Задача коммивояжера¹².
- Руководство по алгоритмам кластеризации¹³.

¹ <http://bit.ly/1a1mgNK>.

² <http://bit.ly/1a1md4B>.

³ <http://bit.ly/1a1m7Ka>.

⁴ <http://bit.ly/2GN6QU5>.

⁵ <http://bit.ly/1a1meWb>.

⁶ <http://bit.ly/1JtgTWJ>.

⁷ <http://bit.ly/2E7vahT>.

⁸ <http://linkd.in/1a1m4ho>.

⁹ <http://linkd.in/1a1Zuj>.

¹⁰ <http://bit.ly/1a1mp3J>.

¹¹ <http://bit.ly/2nNViqS>.

¹² <http://bit.ly/1a1mhkF>.

¹³ <http://bit.ly/1a1mbtp>.

5

Анализ текстовых файлов: определение сходства документов, извлечение словосочетаний и многое другое

Эта глава знакомит с некоторыми фундаментальными понятиями из сферы анализа текста¹ и является своего рода точкой перегиба в этой книге. Мы начали книгу со знакомства с простым частотным анализом данных из Twitter и постепенно дошли до более сложного кластерного анализа данных из профилей LinkedIn. Эта глава, в отличие от предыдущих, начинает обсуждать вопросы обработки и анализа текстовой информации, вводя основы теории поиска информации, такие как TF-IDF, косинусное сходство и определение словосочетаний. Соответственно, она немного сложнее предыдущих глав, поэтому будет полезно прочитать предшествующие главы, прежде чем приниматься за эту.



В предыдущих изданиях этой книги за основу для примеров был взят ныне несуществующий продукт Google+. Но даже притом, что Google+ больше не используется для примеров, основные понятия сохранились и вводятся почти так же, как и раньше. Для преемственности основой для примеров в этой главе по-прежнему служат посты Тима О'Рейли (Tim O'Reilly) в Google+, как и в предыдущих изданиях. Архив этих сообщений поставляется вместе с примерами кода книги в GitHub.

Там, где это возможно, мы не будем изобретать велосипед и писать инструменты анализа с нуля, но сделаем пару «глубоких погружений» в наиболее фундаментальные темы, что совершенно необходимо для понимания идеи анализа текста. Natural Language Toolkit (NLTK) — это мощная технология, о которой вкратце рассказывалось в главе 4; она предоставляет многие из инструментов, которые

¹ Эта книга избегает рассуждений о различиях, которые могут подразумеваться под общими фразами, такими как *анализ текста*, *анализ неструктурированных данных* (Unstructured Data Analytics, UDA) или *поиск информации*, и просто рассматривает их как одно и то же.

мы будем использовать в этой главе. Богатый API этого пакета в первый момент может показаться ошеломляющим, но не волнуйтесь: несмотря на то что анализ текстовой информации является невероятно разнообразной и сложной областью исследований, она основывается на мощном фундаменте, изучение которого может надолго увлечь вас без значительных инвестиций с вашей стороны. Эта и последующие главы направлены на то, чтобы познакомить вас с этими основами. (Полномасштабное введение в NLTK выходит за рамки этой книги, но вы можете ознакомиться с ним в книге *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit* [O'Reilly] на веб-сайте NLTK¹.)



Загрузите последний исправленный исходный код примеров для этой (и любой другой) главы, доступный по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Воспользуйтесь также преимуществами виртуальной машины, описанными в приложении А, чтобы получить максимальное удовольствие от опробования примеров кода.

5.1. Обзор

В этой главе используется небольшой корпус текстов, напоминающих посты, используя который мы начнем наше путешествие в анализ данных на естественном языке. В этой главе вы узнаете:

- о TF-IDF (Term Frequency/Inverse Document Frequency — частота слова/обратная частота документа), фундаментальном методе анализа слов в документах;
- как использовать NLTK для анализа текстов на естественном языке;
- как применить косинусное сходство для решения таких задач, как поиск документов по ключевому слову;
- как извлекать значимые фразы из данных на естественном языке, определяя словосочетания.

5.2. Текстовые файлы

Несмотря на широкую распространенность аудио- и видеоконтента в наши дни, текст продолжает оставаться доминирующей формой коммуникации во всем

¹ <http://bit.ly/1a1mtAk>.

цифровом мире, и ситуация вряд ли изменится в ближайшее время. Развитие даже минимального набора навыков для извлечения значимой статистики из текстовых данных на естественном языке даст вам широкие возможности в решении различных задач, с которыми вы столкнетесь в своей профессиональной карьере при исследовании социальных сетей и других источников информации. В общем случае текстовые данные, доступные через программные интерфейсы социальных сетей, могут быть полноценным HTML-кодом или содержать некоторую базовую разметку, например теги `<br />` и экранированные HTML-сущности. То есть вам определенно потребуется произвести дополнительную фильтрацию для очистки этих данных. Пример 5.1 демонстрирует, как выделить простой текст из поля `content` заметки, реализуя функцию `cleanHtml`. Он использует пакет для работы с HTML с названием `BeautifulSoup`, который преобразует HTML-сущности в обычный текст. Для тех, кто прежде не сталкивался с `BeautifulSoup`, поясню, что это пакет, без которого вы не захотите жить, попробовав его хотя бы раз, — он позволяет обрабатывать разметку HTML, даже недействительную и нарушающую стандарты и другие обоснованные ожидания (что так характерно для веб-данных). Установите пакет командой `pip install beautifulsoup4`, если вы еще не сделали этого.

Пример 5.1. Очистка разметки HTML с удалением тегов и преобразованием HTML-сущностей в простой текст

```
from bs4 import BeautifulSoup # pip install beautifulsoup4

def cleanHtml(html):
    if html == "": return ""

    return BeautifulSoup(html, 'html5lib').get_text()

txt = "Don&#39;t forget about HTML entities and <strong>markup</strong> when "+\
      "mining text!<br />"

print(cleanHtml(txt))
```



Не забывайте, что для получения справки о пакете, классе или методе в терминале очень может пригодиться `rydoc`. Также можно воспользоваться стандартной функцией `help`. Кроме того, в интерпретаторе IPython можно ввести имя метода и добавить в конец вопросительный знак (`?`), чтобы вывести содержимое документирующего комментария.

После очистки разметки HTML с помощью `cleanHtml` получается довольно чистый текст, который, впрочем, можно подвергнуть дополнительной обработке, чтобы убрать оставшийся «мусор». Как вы узнаете в этой и в после-

дующих главах, посвященных анализу текста, удаление мусора из текстовых данных является важной предпосылкой увеличения точности. Вот еще один пример из онлайн-размышлений Тима О'Рейли (Tim O'Reilly) о конфиденциальности.

Вот исходный текст:

```
This is the best piece about privacy that I've read in a long time!  
If it doesn't change how you think about the privacy issue, I'll be  
surprised. It opens:<br /><br />"Many governments (including our own,  
here in the US) would have its citizens believe that privacy is a switch (that  
is, you either reasonably expect it, or you don't). This has been demonstrated  
in many legal tests, and abused in many circumstances ranging from spying  
on electronic mail, to drones in our airspace monitoring the movements of  
private citizens. But privacy doesn't work like a switch - at least it shouldn't  
for a country that recognizes that privacy is an inherent right. In fact,  
privacy, like other components to security, works in layers..."<br /><br />  
Please read!
```

А вот что получилось после его обработки функцией `cleanHtml()`:

```
This is the best piece about privacy that I've read in a long time! If it  
doesn't change how you think about the privacy issue, I'll be surprised. It  
opens: "Many governments (including our own, here in the US) would have its  
citizens believe that privacy is a switch (that is, you either reasonably expect  
it, or you don't). This has been demonstrated in many legal tests, and abused  
in many circumstances ranging from spying on electronic mail, to drones in our  
airspace monitoring the movements of private citizens. But privacy doesn't work  
like a switch - at least it shouldn't for a country that recognizes that privacy  
is an inherent right. In fact, privacy, like other components to security, works  
in layers..." Please read!
```

Манипулирование очищенным текстом, полученным из любой социальной сети или корпуса, собранного любым другим альтернативным способом, является основой для остальных примеров анализа текста в этой главе. В следующем разделе мы познакомимся с одной из классических метрик, используемых для статистического анализа данных на естественном языке.

5.3. Краткое введение в TF-IDF

Даже притом, что такие методы обработки естественного языка (Natural Language Processing, NLP), как сегментация предложений, лексемизация, фрагментация слов и определение сущностей, необходимы для достижения более

глубокого понимания текстовых данных, для начала полезно познакомиться с некоторыми основами из теории поиска информации. Поэтому в оставшейся части этой главы мы рассмотрим некоторые из ее наиболее фундаментальных аспектов, включая TF-IDF и метрику косинусного сходства, а также коснемся теории выделения словосочетаний. В главе 6 мы затронем более глубокие вопросы NLP, продолжив эту дискуссию.



Если вы хотите более подробно ознакомиться с теорией информационного поиска, то советую книгу Кристофера Маннинга (Christopher Manning), Прабхакара Рагхавана (Prabhakar Raghavan) и Хайнриха Шютце (Hinrich Schütze) *Introduction to Information Retrieval*¹ (Cambridge University Press), свободно доступную в интернете² и содержащую больше информации, чем вам (возможно) хотелось бы узнать об этой области.

Информационный поиск — обширная область со множеством разделов. Эта наша дискуссия сводится к обсуждению TF-IDF, одного из фундаментальных методов извлечения релевантных документов из корпуса (коллекции). Аббревиатура TF-IDF расшифровывается как *term frequency* – *inverse document frequency* (частота слова — обратная частота документа) и обозначает метрику, которую можно использовать для поиска документов в корпусе путем вычисления нормализованных оценок, выражающих относительную важность слов в документах.

Математически TF-IDF выражается как произведение частоты слова на обратную частоту документа, $tf_idf = tf \times idf$, где tf представляет важность слова в конкретном документе, а idf — важность слова для всего корпуса. Их произведение дает оценку, учитывающую оба фактора, которая являлась неотъемлемой частью любой крупной поисковой системы в какой-то момент ее существования. Чтобы получить более полное представление о смысле оценки TF-IDF, рассмотрим вычисления, участвующие в расчетах общей оценки.

5.3.1. Частота слова

Для простоты допустим, что у нас есть корпус с тремя документами, а слова определяются по пробельным символам, как показано в примере 5.2 с обычным кодом на Python.

¹ Кристофер Д. Маннинг, Прабхакар Рагхаван, Хайнрих Шютце. Введение в информационный поиск. М.: Вильямс, 2014. ISBN 978-5-8459-1623-5, 978-0-5218-6571-5. — *Примеч.* *пер.*

² <http://stanford.io/1a1mAvP>.

Пример 5.2. Структуры данных, используемые в остальных примерах в этой главе

```
corpus = {
  'a' : "Mr. Green killed Colonel Mustard in the study with the candlestick. \
Mr. Green is not a very nice fellow.",
  'b' : "Professor Plum has a green plant in his study.",
  'c' : "Miss Scarlett watered Professor Plum's green plant while he was away \
from his office last week."
}
terms = {
  'a' : [ i.lower() for i in corpus['a'].split() ],
  'b' : [ i.lower() for i in corpus['b'].split() ],
  'c' : [ i.lower() for i in corpus['c'].split() ]
}
```

Частоту слова можно определить просто как количество раз, которое это слово встречается в тексте, но чаще она нормализуется общим количеством слов в тексте, чтобы учитывалась длина документа. Например, слово «green» (после нормализации в нижний регистр) дважды встречается в `corpus['a']` и только один раз в `corpus['b']`, поэтому `corpus['a']` получит более высокую оценку, если единственным критерием будет служить частота. Но если выполнить нормализацию по длине документов, `corpus['b']` будет оценен по частоте слова «green» немного выше (1/9), чем `corpus['a']` (2/19), потому что `corpus['b']` короче, чем `corpus['a']`. Часто при обработке составных запросов, таких как «Mr.Green», оценки частот каждого слова из запроса в каждом документе суммируются и в ответ возвращаются документы, ранжированные по суммарной оценке частоты слова.

Для иллюстрации в табл. 5.1 перечислены документы из нашего корпуса и их нормализованные оценки частоты слов для запроса «Mr. Green».

Таблица 5.1. Оценки частоты слов для запроса «Mr. Green»

Документ	tf(mr.)	tf(green)	Сумма
corpus['a']	2/19	2/19	4/19 (0,2105)
corpus['b']	0	1/9	1/9 (0,1111)
corpus['c']	0	1/16	1/16 (0,0625)

В этом искусственном примере накопленная оценка частоты слова дает правильный результат и помогает вернуть `corpus['a']` (ожидаемый документ), потому что `corpus['a']` — единственный документ, который содержит составную лексику «Mr. Green». Однако такой подход может породить несколько проблем

из-за того, что модель оценки частоты слова рассматривает каждый документ как неупорядоченный набор слов. Например, в ответ на запросы «Green Mr.» и «Green Mr. Foo» мы получили бы тот же результат, что и для запроса «Mr. Green», хотя ни одной из этих фраз нет ни в одном из документов. Более того, легко можно придумать ситуации, когда метод ранжирования по частоте слов дает плохие результаты из-за ненадлежащей обработки знаков препинания и из-за того, что не учитывается контекст, окружающий лексемы.

Использование одних только частот слов часто является причиной неправильной оценки релевантности документов, потому что при этом не учитываются так называемые *стоп-слова*¹ — распространенные слова, часто встречающиеся во многих документах. Иначе говоря, все слова получают одинаковый вес, независимо от их фактического значения. Например, словосочетание «the green plant» (зеленые насаждения) содержит стоп-слово «the», которое искажает общую оценку в пользу corpus['a'], потому что «the» появляется в этом документе дважды, как и слово «green». Напротив, в corpus['c'] слова «green» и «plant» появляются только один раз.

Соответственно документы в корпусе получают оценки, указанные в табл. 5.2. То есть документ corpus['a'] будет признан более релевантным, чем corpus['c'], хотя интуиция подсказывает, что правильный результат должен выглядеть несколько иначе. (К счастью, corpus['b'] получил самый высокий ранг.)

Таблица 5.2. Оценки частоты слов для запроса «the green plant»

Документ	tf(the)	tf(green)	tf(plant)	Сумма
corpus['a']	2/19	2/19	0	4/19 (0,2105)
corpus['b']	0	1/9	1/9	2/9 (0,2222)
corpus['c']	0	1/16	1/16	1/8 (0,125)

5.3.2. Обратная частота документа

Наборы инструментов, такие как NLTK, предоставляют списки стоп-слов, которые можно использовать для фильтрации таких терминов, как *and*, *a* и *the*,

¹ Стоп-слова — это слова, которые часто встречаются в тексте, но почти не передают информации. Типичными примерами стоп-слов в английском языке могут служить *a*, *an*, *the* и другие артикли. (Примерами стоп-слов в русском языке могут служить союзы, предлоги, местоимения и пр. — *Примеч. пер.*)

но имейте в виду, что есть слова, отсутствующие в самых полных списках стоп-слов, но все же довольно распространенные в некоторых областях. Конечно, вы можете определить свой список стоп-слов, учитывающий особенности своей предметной области, однако есть еще одна частотная метрика — обратная частота документа, которая предоставляет обобщенную возможность нормализации оценок документов в корпусе. Она учитывает появление распространенных слов в наборе документов и общее количество документов, в которых присутствуют слова из запроса.

Эта метрика получает более высокое значение, если слово несколько необычно для всего корпуса, что помогает немного ослабить проблему со стоп-словами, которую мы только что рассмотрели. Например, запрос «green» для нашего корпуса документов получит более низкую оценку обратной частоты документа, чем запрос «candlestick» (свеча), потому что слово «green» встречается во всех документах, а слово «candlestick» — только в одном. Есть один интересный математический нюанс, касающийся вычисления обратной частоты документа, — на практике вместо обратной частоты непосредственно используется ее логарифм, чтобы привести получаемые значения к более узкому диапазону, потому что обычно они используются для умножения на частоту слова, то есть как коэффициенты масштабирования. Для справки на рис. 5.1 показана функция логарифма: как видите, она растет все медленнее с увеличением значений в области ее определения, эффективно «сжимая» входные значения.

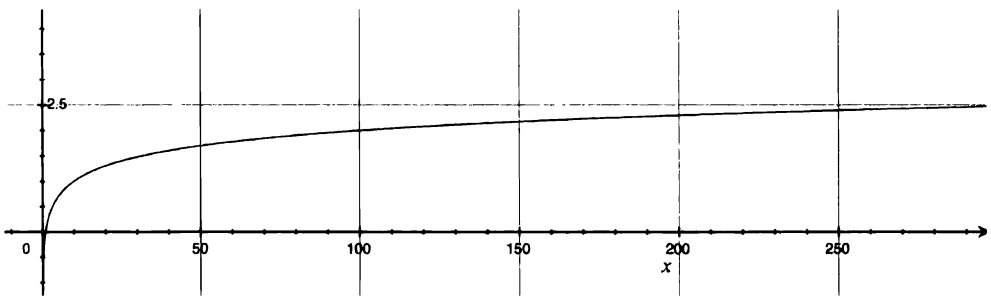


Рис. 5.1. Функция логарифма «сжимает» большой диапазон значений в более компактное пространство — обратите внимание, как все медленнее и медленнее растут значения y с ростом x

В табл. 5.3 приводятся оценки обратной частоты документа, соответствующие оценкам частот слов из предыдущего раздела. Порядок вычисления этих оценок демонстрирует пример 5.3 в следующем разделе. А пока можете рассматривать

оценку IDF, как логарифм частного, получаемого делением количества документов в корпусе на количество документов, содержащих заданное слово. Просматривая эти таблицы, имейте в виду, что оценка частоты слова вычисляется для каждого документа в отдельности, а оценка обратной частоты документа — для всего корпуса. В этом есть определенный смысл, учитывая, что оценка обратной частоты документа служит для нормализации распространенных слов во всем корпусе.

Таблица 5.3. Оценки обратной частоты документа для запросов «mr. green» и «the green plant»

idf(mr.)	idf(green)	idf(the)	idf(plant)
$1 + \log(3/1) = 2,0986$	$1 + \log(3/3) = 1,0$	$1 + \log(3/3) = 1,0$	$1 + \log(3/2) = 1,4055$

5.3.3. TF-IDF

Итак, мы замкнули круг и теперь у нас есть все для оценки релевантности документов запросу, состоящему из нескольких слов, с учетом частоты появления слов в документах, размеров документов и обобщенной уникальности слов для всего корпуса. Мы можем объединить оценки частоты слова и обратной частоты документа, перемножив их, то есть найти оценку $TF-IDF = TF \times IDF$. В примере 5.3 представлена простейшая реализация вычисления этой оценки, которая должна помочь закрепить понимание описанного выше. Просмотрите его, а затем мы обсудим несколько примеров запросов.

Пример 5.3. Вычисление оценки TF-IDF

```
from math import log

# Укажите в запросе слова, присутствующие в корпусе
QUERY_TERMS = ['mr.', 'green']

def tf(term, doc, normalize=True):
    doc = doc.lower().split()
    if normalize:
        return doc.count(term.lower()) / float(len(doc))
    else:
        return doc.count(term.lower()) / 1.0

def idf(term, corpus):
```

```

num_texts_with_term = len([True for text in corpus if term.lower()
                           in text.lower().split()])

# Оценка tf-idf вычисляется умножением idf на значение tf, которое
# меньше 0, но чтобы получить непротиворечивые оценки, необходимо
# вернуть число больше 1. (Произведение двух чисел меньше 1 всегда меньше
# каждого из них.)

try:
    return 1.0 + log(float(len(corpus)) / num_texts_with_term)
except ZeroDivisionError:
    return 1.0

def tf_idf(term, doc, corpus):
    return tf(term, doc) * idf(term, corpus)

corpus = \
    {'a': 'Mr. Green killed Colonel Mustard in the study with the candlestick. \
Mr. Green is not a very nice fellow.',
     'b': 'Professor Plum has a green plant in his study.',
     'c': "Miss Scarlett watered Professor Plum's green plant while he was away \
from his office last week."}

for (k, v) in sorted(corpus.items()):
    print(k, ': ', v)
print()

# Найти оценку для всего запроса, вычислив накопленные оценки tf_idf
# для каждого слова в запросе

query_scores = {'a': 0, 'b': 0, 'c': 0}
for term in [t.lower() for t in QUERY_TERMS]:
    for doc in sorted(corpus):
        print('TF({0}): {1}'.format(doc, term), tf(term, corpus[doc]))
        print('IDF: {0}'.format(term), idf(term, corpus.values()))
        print()

    for doc in sorted(corpus):
        score = tf_idf(term, corpus[doc], corpus.values())
        print('TF-IDF({0}): {1}'.format(doc, term), score)
        query_scores[doc] += score
    print()

print("Overall TF-IDF scores for query '{0}'".format(' '.join(QUERY_TERMS)))
for (doc, score) in sorted(query_scores.items()):
    print(doc, score)

```

Вот как выглядит результат выполнения этого примера:

```
a : Mr. Green killed Colonel Mustard in the study...
b : Professor Plum has a green plant in his study.
c : Miss Scarlett watered Professor Plum's green...
```

```
TF(a): mr. 0.105263157895
TF(b): mr. 0.0
TF(c): mr. 0.0
IDF: mr. 2.09861228867
```

```
TF-IDF(a): mr. 0.220906556702
TF-IDF(b): mr. 0.0
TF-IDF(c): mr. 0.0
```

```
TF(a): green 0.105263157895
TF(b): green 0.111111111111
TF(c): green 0.0625
IDF: green 1.0
```

```
TF-IDF(a): green 0.105263157895
TF-IDF(b): green 0.111111111111
TF-IDF(c): green 0.0625
```

```
Overall TF-IDF scores for query 'mr. green'
a 0.326169714597
b 0.111111111111
c 0.0625
```

Несмотря на то что мы работаем с очень маленьким набором данных, вычисления здесь используются те же самые, что и при работе с большими наборами. В табл. 5.4 приводятся консолидированные результаты оценки трех запросов, включающих четыре разных слова:

- green;
- mr. green;
- the green plant.

Несмотря на то что оценки IDF для слов вычисляются на основе всего корпуса, они повторно приводятся для каждого документа, чтобы проще было проверить вычисление TF-IDF, просматривая одну строку и умножая два числа. Изучая результаты оценки запросов, обратите внимание, насколько мощной является оценка TF-IDF, даже притом, что она не учитывает близость или порядок слов в документе.

Таблица 5.4. Вычисления, связанные с оценкой TF-IDF и реализованные в примере 5.3

Документ	tf(mr.)	tf(green)	tf(the)	tf(plant)
corpus['a']	0,1053	0,1053	0,1053	0
corpus['b']	0	0,1111	0	0,1111
corpus['c']	0	0,0625	0	0,0625

idf(mr.)	idf(green)	idf(the)	idf(plant)
2,0986	1,0	2,099	1,4055

Документ	tf-idf(mr.)	tf-idf (green)	tf-idf (the)	tf-idf (plant)
corpus['a']	$0,1053 \times 2,0986 = 0,2209$	$0,1053 \times 1,0 = 0,1053$	$0,1053 \times 2,099 = 0,2209$	$0 \times 1,4055 = 0$
corpus['b']	$0 \times 2,0986 = 0$	$0,1111 \times 1,0 = 0,1111$	$0 \times 2,099 = 0$	$0,1111 \times 1,4055 = 0,1562$
corpus['c']	$0 \times 2,0986 = 0$	$0,0625 \times 1,0 = 0,0625$	$0 \times 2,099 = 0$	$0,0625 \times 1,4055 = 0,0878$

Те же результаты для каждого запроса показаны в табл. 5.5, где значения TF-IDF суммированы по документам.

Таблица 5.5. Суммированные значения TF-IDF, вычисленные в примере 5.3 (жирным выделены оценки с наибольшим значением для каждого из трех запросов)

Запрос	corpus['a']	corpus['b']	corpus['c']
green	0,1053	0.1111	0,0625
Mr. Green	$0,2209 + 0,1053 =$ 0,3262	$0 + 0,1111 = 0,1111$	$0 + 0,0625 = 0,0625$
the green plant	$0,2209 + 0,1053 + 0 =$ 0,3262	$0 + 0,1111 + 0,1562 =$ $0,2673$	$0 + 0,0625 + 0,0878 =$ $0,1503$

Мы получили вполне приемлемые результаты, если смотреть с точки зрения качества. Документ corpus['b'] признан наиболее релевантным для запроса «green», corpus['a'] лишь чуть-чуть отстает от него. В данном случае решающим фактором стала длина corpus['b'], существенно меньшая, чем длина corpus['a'], из-за чего нормализованная оценка TF для единственного вхождения слова «green» в corpus['b'] оказалась выше оценки для двух вхождений

«Green» в corpus['a']. Слово «green» присутствует во всех трех документах, чистый эффект влияния оценки IDF оказался нулевым.

Обратите внимание, что если бы мы вернули 0,0 вместо 1,0 для слова «green», как это делается в некоторых реализациях IDF, тогда оценка TF-IDF для «green» получилась бы равной 0,0 для всех трех документов из-за эффекта умножения значения TF на ноль. Иногда, в зависимости от конкретной ситуации, предпочтительнее вернуть 0,0 для IDF, а не 1,0. Например, если у вас есть 100 000 документов и слово «green» присутствует в каждом из них, вы почти наверняка посчитаете его стоп-словом и захотите полностью исключить его влияние на оценку релевантности запросу.

Для запроса «Mr. Green» очевидным победителем является документ corpus['a']. Этот же документ получает наибольшую оценку для запроса «the green plant». Попробуйте сами объяснить, почему corpus['a'] получил более высокую оценку для этого запроса, чем corpus['b'], который на первый взгляд выглядит более релевантным.

И наконец, обратите внимание, что реализация, представленная в примере 5.3, корректирует оценку IDF, прибавляя 1,0 к вычисленному логарифму. Это сделано для иллюстрации, а также потому, что мы имеем дело с тривиальным набором документов. Без такой корректировки функция `idf` может возвращать значения меньше 1,0, из-за чего в вычислении TF-IDF мы пришли бы к умножению двух дробей. Поскольку произведение двух дробей всегда меньше любого из сомножителей, легко упустить из виду крайний случай в вычислении TF-IDF. Согласно идее, лежащей в основе расчетов TF-IDF, мы должны перемножать оценки TF и IDA так, чтобы неизменно получать оценки TF-IDF, значений которых больше для релевантных документов и меньше для нерелевантных.

5.4. Оценка запросов данных на естественном языке с использованием TF-IDF

Давайте возьмем теорию, исследованную в предыдущем разделе, и применим ее на практике. В этом разделе вы официально познакомитесь с NLTK, мощным набором инструментов для обработки естественного языка, и используете его для анализа данных на естественном языке.

5.4.1. Введение в Natural Language Toolkit

Если вы еще не сделали этого, установите пакет Natural Language Toolkit (NLTK) для Python, используя команду `pip install nltk`. Пакет NLTK написан так, чтобы легко можно было приступить к исследованию данных и сформировать некоторые представления без больших первоначальных инвестиций. Прежде чем двинуться дальше, попробуйте повторить сеанс интерпретатора, представленный в примере 5.4, чтобы познакомиться с некоторыми мощными возможностями NLTK. Если прежде вы не были знакомы с NLTK, не забывайте, что можете использовать встроенную функцию `help` для получения дополнительной информации, когда она потребуется. Например, вызов `help(nltk)` в сеансе интерпретатора выведет документацию с описанием пакета NLTK.

Не все функциональные возможности NLTK предназначены для включения в промышленное программное обеспечение, потому что выходные данные выводятся в консоль и не могут быть включены в какую-нибудь структуру данных, например список. В связи с этим такие методы, как `nltk.text.concordance`, считаются «демонстрационными». Кстати говоря, многие модули в пакете NLTK имеют функцию `demo`, которую можно вызвать, чтобы получить некоторое представление о том, как использовать возможности, предлагаемые этими модулями, а их исходный код может служить отличной отправной точкой для изучения приемов использования новых API. Например, можно вызвать `nltk.text.demo()` в интерпретаторе, чтобы получить дополнительную информацию о возможностях модуля `nltk.text`.

Пример 5.4 демонстрирует некоторые отправные точки в процессе исследования данных с примерами результатов, включенных как часть интерактивного сеанса интерпретатора. Те же команды включены в исходный код блокнота Jupyter Notebook для этой главы. Попробуйте выполнить этот пример у себя и изучите результаты, получаемые на каждом шаге. Сможете ли вы самостоятельно понять, что делают команды, выполняемые в потоке сеанса? Посмотрите, а затем мы обсудим некоторые детали.



Следующий пример включает стоп-слова, которые, как отмечалось ранее, часто встречаются в тексте, но обычно не несут полезной информации (например, *a*, *an*, *the* и др.).

Пример 5.4. Исследование данных с помощью NLTK

```
# Демонстрация некоторых возможностей NLTK в исследовании данных.
# Здесь вы найдете подсказки, которые пригодятся в интерактивном сеансе.

import json
import nltk

# Загрузить вспомогательные пакеты nltk, если они еще не были установлены
nltk.download('stopwords')

# Загрузить текстовые данные на естественном языке
DATA = 'resources/ch05-textfiles/ch05-timoreilly.json'
data = json.loads(open(DATA).read())

# Объединить заголовки и содержимое сообщений
all_content = " ".join([ i['title'] + " " + i['content'] for i in data ])

# Вывести размер текстовых данных в байтах
print(len(all_content))

tokens = all_content.split()
text = nltk.Text(tokens)

# Примеры вхождений слова "open"
text.concordance("open")

# Часто встречающиеся словосочетания в тексте (обычно осмысленные фразы)
text.collocations()

# Частотный анализ интересующих слов
fdist = text.vocab()
print(fdist["open"])
print(fdist["source"])
print(fdist["web"])
print(fdist["2.0"])

# Количество слов в тексте
print('Number of tokens:', len(tokens))

# Количество уникальных слов в тексте
print('Number of unique words:', len(fdist.keys()))

# Часто встречающиеся слова, не являющиеся стоп-словами
print('Common words that aren\'t stopwords')
print([w for w in list(fdist.keys())[:100]
       if w.lower() not in nltk.corpus.stopwords.words('english')])

# Длинные слова, не являющиеся адресами URL
print('Long words that aren\'t URLs')
```

```
print([w for w in fdist.keys() if len(w) > 15 and 'http' not in w])

# Количество адресов URL
print('Number of URLs: ', len([w for w in fdist.keys() if 'http' in w]))

# Топ-10 наиболее часто встречающихся слов
print('Top 10 Most Common Words')
print(fdist.most_common(10))
```



В примерах, представленных в этой главе, включая предыдущий, лексемизация текста выполняется с помощью метода `split`. Но вообще процедура лексемизации несколько сложнее, чем разбиение текста по пробельным символам. В главе 6 мы познакомимся с более сложными методами лексемизации, которые лучше подходят для общего случая.

Последняя команда в сеансе выше выводит список слов, отсортированных по частоте. Неудивительно, что чаще других в тексте встречаются такие стоп-слова, как *to* и *of*, но при этом наблюдается быстрое снижение частот и распределение имеет очень длинный хвост. В данном примере использована небольшая выборка текстовых данных, тем не менее это свойство проявляется в частотном анализе любых текстов на естественном языке.

Закон Ципфа (Zipf)¹, известный эмпирический закон частотного распределения слов в текстах на естественных языках, утверждает, что частота слова в корпусе обратно пропорциональна его порядковому номеру в таблице частот. То есть если на долю наиболее часто встречающегося слова в корпусе приходится $N\%$ от общего количества слов, то на долю второго наиболее часто встречающегося слова в корпусе должно приходиться $(N/2)\%$ слов, на долю третьего наиболее часто встречающегося слова $(N/3)\%$ и т. д. На графике такое распределение (даже для небольшой выборки) имеет форму кривой, изображенной на рис. 5.2.

Хотя на первый взгляд это неочевидно, но большая часть области в таком распределении лежит в его хвосте, и для достаточно большого корпуса, охватывающего представительную выборку языка, хвост всегда получается довольно длинным. Если попробовать построить график такого распределения с осями, масштабированными по логарифмам, для репрезентативного размера выборки кривая получится близкой к прямой линии.

Закон Ципфа помогает получить представление о том, как должно выглядеть частотное распределение слов в корпусе, и задает некоторые эмпирические пра-

¹ <http://bit.ly/1a1mCUD>.

вила, которые могут пригодиться при оценке частот. Например, если известно, что корпус содержит миллион (неуникальных) слов, и предполагается, что на долю наиболее часто используемого слова (обычно *the* в английском языке) приходится 7% слов¹, вы можете вывести общее число логических операций, выполняемых алгоритмом для конкретного среза в частотном распределении. Иногда такой простой арифметики достаточно, чтобы оценить продолжительность вычислений или убедиться в осуществимости некоторых вычислений на достаточно большом наборе данных за разумный срок.

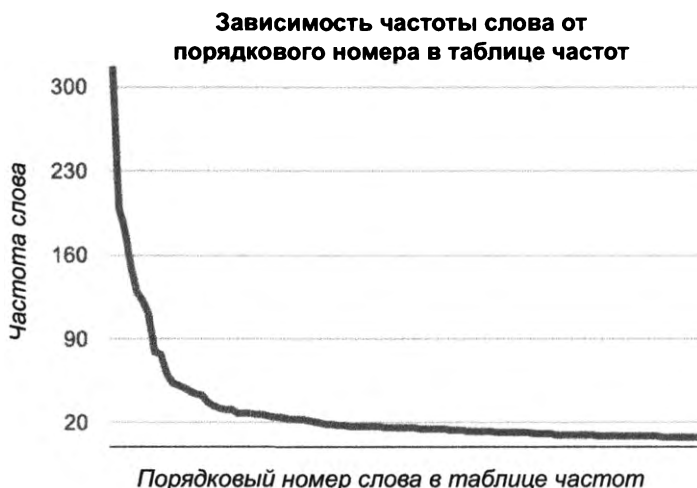


Рис. 5.2. График частотного распределения слов в небольшой выборке данных простирается близко к обеим осям; если построить этот же график в логарифмическом масштабе по обеим осям, он получится близким к прямой линии с отрицательным наклоном



Попробуйте изобразить такую же кривую, как на рис. 5.2, для своего небольшого текстового корпуса, используя методы, представленные в этой главе, и средства построения графиков в IPython, описанные в главе 1.

¹ На слово *the* приходится 7% лексем в корпусе Brown Corpus (<http://bit.ly/1a1mB2X>), и это число может служить разумной отправной точкой, если вы ничего не знаете об исследуемом вами корпусе.

5.4.2. Вычисление оценки TF-IDF для текста на естественном языке

Давайте попробуем вычислить оценку TF-IDF для текстовой выборки и посмотрим, как ее можно использовать для определения релевантности документов запросу. В пакете NLTK имеются некоторые готовые абстракции, которые можно использовать вместо своих собственных, что поможет нам почти без труда понять основную суть теории. В примере 5.5 используется выборка текстовых данных, распространяемая с исходным кодом для этой главы в виде файла JSON. С его помощью вы сможете выполнить несколько запросов и оценить документы по релевантности.

Пример 5.5. Вычисление оценки TF-IDF для текстовых данных

```
import json
import nltk

# Определите здесь свой запрос
QUERY_TERMS = ['Government']

# Загрузить данные на естественном языке из указанного файла
DATA = 'resources/ch05-textfiles/ch05-timoreilly.json'
data = json.loads(open(DATA).read())

activities = [post['content'].lower().split()
               for post in data
               if post['content'] != ""]

# TextCollection определяет абстракции tf, idf и tf_idf,
# поэтому нам не требуется определять свои версии

tc = nltk.TextCollection(activities)

relevant_activities = []

for idx in range(len(activities)):
    score = 0
    for term in [t.lower() for t in QUERY_TERMS]:
        score += tc.tf_idf(term, activities[idx])
    if score > 0:
        relevant_activities.append({'score': score, 'title': data[idx]['title']})

# Сортировать результаты по релевантности и вывести их

relevant_activities = sorted(relevant_activities,
                             key=lambda p: p['score'], reverse=True)
```

```
for activity in relevant_activities:
    print('Title: {}'.format(activity['title']))
    print('Score: {}'.format(activity['score']))
    print()
```

Примеры оценки релевантности некоторых рассуждений Тима О'Рейли (Tim O'Reilly) запросу «Government»:

```
Title: Totally hilarious and spot-on. Has to be the best public service video...
Score: 0.106601312641
```

```
Title: Excellent set of principles for digital government. Echoes those put...
Score: 0.102501262155
```

```
Title: "We need to show every American competent government services they can...
Score: 0.0951797434292
```

```
Title: If you're interested about the emerging startup ecosystem around...
Score: 0.091897683311
```

```
Title: I'm proud to be a judge for the new +Code for America tech awards. If...
Score: 0.0873781251154
...
```

Возможность ранжировать контент по релевантности заданному запросу дает огромное преимущество при анализе неструктурированных текстовых данных. Попробуйте выполнить несколько других запросов и проверьте, насколько хорошо работает метрика TF-IDF, не забывая при этом, что абсолютные значения оценок на самом деле не важны — они просто дают возможность найти и отсортировать документы по релевантности. Затем поразмышляйте о бесчисленных способах настройки или усовершенствования этой метрики, чтобы сделать ее еще более эффективной. Одно очевидное улучшение, которое мы оставляем читателю в качестве самостоятельного упражнения, заключается в том, чтобы привести глаголы к общему основанию, устранив различия в их написании, обусловленные спряжениями, числами и лицами, и сделать вычисления более точными. Некоторые простые в использовании реализации распространенных алгоритмов стемминга вы найдете в модуле `nlk.stem`.

Теперь возьмем наши новые инструменты и применим их для решения основной задачи — поиска похожих документов. В конце концов, как только вы выделите интересующий документ, следующим естественным шагом является поиск других документов, которые могут представлять интерес.

5.4.3. Поиск похожих документов

После обработки запроса и обнаружения релевантных документов следующим естественным шагом является определение сходства документов. Если для выборки из корпуса документов, соответствующих поисковому запросу, можно использовать оценку TF-IDF, то для определения сходства документов можно использовать оценку косинусного сходства, которая является одним из наиболее распространенных методов сравнения документов друг с другом. Для понимания идеи косинусного сходства необходимо кратко познакомиться с моделью векторного пространства, чем мы и займемся в следующем разделе.

Теория модели векторного пространства и косинусного сходства

Как уже подчеркивалось, метод TF-IDF представляет документы как неупорядоченные наборы слов. Другим удобным способом представления документов является модель, называемая *векторным пространством*. Суть теории, лежащей в основе модели векторного пространства, заключается в наличии большого многомерного пространства, в котором каждому документу соответствует свой вектор, а расстояние между любыми двумя векторами определяет степень сходства соответствующих им документов. Одна из самых замечательных особенностей модели векторных пространств заключается в возможности представить запрос в виде вектора и обнаружить наиболее релевантные документы, определив векторы документов с наименьшим расстоянием до вектора запроса.

Дать более или менее исчерпывающее описание теории в таком коротком разделе практически невозможно, однако очень важно иметь хотя бы упрощенное представление о модели векторного пространства, если вас интересует проблема анализа текста или информационного поиска. Если теоретические основы вам не интересны и вы хотели бы сразу перейти к деталям реализации, можете смело пропустить этот раздел.



Этот раздел предполагает знание основ тригонометрии. Если ваши знания из школьного курса математики потускнели, рассматривайте этот раздел как отличную возможность освежить память. Если вы не настроены на изучение теории, просто просмотрите этот раздел и помните, что вычисление подобия, которое мы будем использовать для поиска похожих документов, основано на строгой математической теории.

Для начала уточним, что подразумевается под термином *вектор*, потому что в разных областях его толкование имеет множество тонких вариаций. Вообще говоря, вектор — это список чисел, выражающий направление относительно начала координат и расстояние от этого начала. Вектор можно представить в виде отрезка, соединяющего начало координат и точку в N -мерном пространстве.

Для иллюстрации представьте документ, который определяется только двумя словами («Open» и «Web») с соответствующим вектором $(0.45, 0.67)$, элементами которого являются такие значения, как оценки TF-IDF для слов. В векторном пространстве этот документ можно представить отрезком в двумерной системе координат, соединяющим начало координат $(0, 0)$ и точку $(0.45, 0.67)$. В координатной плоскости X/Y ось X будет представлять слово «Open», ось Y — слово «Web», а вектор от $(0, 0)$ до $(0.45, 0.67)$ — рассматриваемый документ. Нетривиальные документы обычно содержат сотни слов, но для их моделирования в многомерных пространствах применяются те же принципы; просто такие пространства сложнее представить визуально.

Попробуйте вместо документа, представленного вектором в двумерном пространстве, вообразить документ, представленный тремя измерениями, такими как «Open», «Web» и «Government». Затем сделайте еще шаг, пусть и трудный, и вообразите вектор в пространстве с большим числом размерностей, которые невозможно нарисовать или увидеть. Если вы смогли это сделать, у вас не должно быть проблем с уверенностью, что векторные операции, применимые к двумерному пространству, точно так же можно применить к 10-мерному или 367-мерному пространству. На рис. 5.3 показан пример вектора в трехмерном пространстве.

Учитывая, что документы можно моделировать в виде векторов в пространстве слов, каждый член которых представлен соответствующей оценкой TF-IDF, задача сводится к тому, чтобы определить метрику, наилучшим образом представляющую сходство между двумя документами. Как оказывается, для сравнения векторов с успехом можно использовать косинус угла между ними, и называется эта метрика *косинусным сходством* векторов. Хотя это и неочевидно, но годы научных исследований показали, что вычисление косинусного сходства документов, представленных в виде векторов, является очень эффективной метрикой. (Однако она страдает от множества тех же проблем, что и TF-IDF, краткое перечисление которых вы найдете в разделе «Заключительные замечания».) Строгое доказательство деталей, лежащих в основе метрики косинусного сходства, выходит за рамки этой книги, но суть в том, что косинус угла между двумя векторами указывает на сходство

между ними и определяется как скалярное произведение¹ соответствующих единичных векторов.

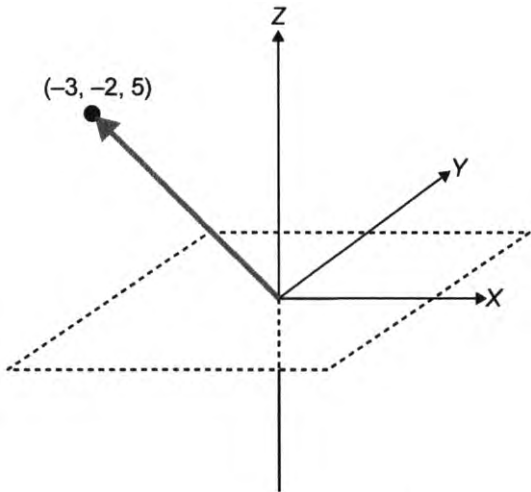


Рис. 5.3. Пример вектора с элементами (-3, -2, 5) в трехмерном пространстве; чтобы попасть в точку с этими координатами, нужно отложить три единицы влево, две единицы назад и пять единиц вверх

Важно понять, что чем ближе два вектора друг к другу, тем меньше угол между ними и, следовательно, тем больше косинус угла между ними. Два одинаковых вектора будут иметь угол 0 градусов и метрику сходства 1,0, в то время как два ортогональных вектора будут иметь угол 90 градусов и метрику сходства 0,0. Это демонстрирует следующая таблица:

$\overline{doc1} \times \overline{doc2} = \ doc1\ \times \ doc2\ \times \cos \Theta$	Дано (в тригонометрии)
$\frac{\overline{doc1} \times \overline{doc2}}{\ doc1\ \times \ doc2\ } = \cos \Theta$	Группировка
$\widehat{doc1} \times \widehat{doc2} = \cos \Theta$	Для «единичных векторов»
$\widehat{doc1} \times \widehat{doc2} = \text{Similarity}(doc1, doc2)$	Подстановка предполагается: $\cos \Theta = \text{Similarity}(doc1, doc2)$

¹ <http://bit.ly/1a1mBjn>.

Напомню, что единичный вектор имеет длину 1,0 (по определению), вы можете видеть, что красота вычисления сходства документов с использованием единичных векторов заключается в том, что они уже нормализованы по длине. Все эти знания мы используем в следующем разделе.

Кластеризация сообщений с использованием косинусного сходства

Один из наиболее важных аспектов, который следует усвоить из предыдущего обсуждения, состоит в том, что *для вычисления сходства двух документов достаточно создать для каждого вектор слов и вычислить скалярное произведение соответствующих им единичных векторов*. К счастью, в NLTK имеется функция `nltk.cluster.util.cosine_distance(v1,v2)`, вычисляющая косинусное сходство, благодаря которой сравнивать документы довольно просто. Как показано в примере 5.6, основная наша работа заключается в создании соответствующих векторов слов. Этот пример вычисляет векторы для данной пары документов, присваивая их элементам оценки TF-IDF. Поскольку словари двух документов едва ли будут идентичными, в каждом векторе оставлены заполнители со значением 0,0, которые представляют слова, отсутствующие в одном документе, но присутствующие в другом. В результате мы получаем два вектора одинаковой длины и с одинаковым порядком следования элементов, которые можно использовать в векторных операциях.

Например, предположим, что *document1* содержит слова (*A, B, C*) и ему соответствует вектор весов TF-IDF (*0.10, 0.15, 0.12*), а *document2* содержит слова (*C, D, E*) и ему соответствует вектор весов TF-IDF (*0.05, 0.10, 0.09*). Тогда производный вектор для *document1* будет иметь вид (*0.10, 0.15, 0.12, 0.0, 0.0*), а для *document2* — (*0.0, 0.0, 0.05, 0.10, 0.09*). Оба этих вектора можно передать в функцию `cosine_distance`, которая вычислит и вернет косинусное сходство. Внутренне `cosine_distance` использует модуль `numpy`, который обеспечивает очень эффективное вычисление скалярного произведения единичных векторов.



Код в этом разделе повторно использует метрику TF-IDF, представленную ранее, но вообще в роли оценки можно использовать любую другую метрику. Однако TF-IDF (или его разновидности) широко используется во многих реализациях и может служить прекрасной отправной точкой.

Пример 5.6 иллюстрирует применение косинусного сходства для поиска документов, наиболее похожих на каждый документ в корпусе. Этот подход с успе-

хом можно применить к любым другим текстовым данным на естественном языке, таким как статьи в блогах или книги.

Пример 5.6. Поиск похожих документов с использованием метрики косинусного сходства

```
import json
import nltk
import nltk.cluster

# Загрузить текстовые данные на естественном языке
DATA = 'resources/ch05-textfiles/ch05-timoreilly.json'
data = json.loads(open(DATA).read())

all_posts = [ (i['title'] + " " + i['content']).lower().split() for i in data ]

# TextCollection определяет абстракции tf, idf и tf-idf для получения оценок

tc = nltk.TextCollection(all_posts)

# Вычислить матрицу слово/документ, чтобы обращение к
# td_matrix[doc_title][term] возвращало оценку tf-idf для слова в документе

td_matrix = {}
for idx in range(len(all_posts)):
    post = all_posts[idx]
    fdist = nltk.FreqDist(post)

    doc_title = data[idx]['title'].replace('\n', '')
    td_matrix[doc_title] = {}

    for term in fdist.keys():
        td_matrix[doc_title][term] = tc.tf_idf(term, post)

# Сконструировать векторы с оценками для одинаковых слов
# в одних и тех же позициях...

distances = {}
for title1 in td_matrix.keys():

    distances[title1] = {}
    (min_dist, most_similar) = (1.0, ('', ''))

    for title2 in td_matrix.keys():

        # Мы не должны изменять оригинальные структуры данных,
        # потому что они не раз понадобятся в этом цикле

        terms1 = td_matrix[title1].copy()
```

```

terms2 = td_matrix[title2].copy()

# Заполнить "пробелы" в каждом словаре, чтобы получить
# векторы одинаковой длины
for term1 in terms1:
    if term1 not in terms2:
        terms2[term1] = 0

for term2 in terms2:
    if term2 not in terms1:
        terms1[term2] = 0

# Создать векторы из словарей
v1 = [score for (term, score) in sorted(terms1.items())]
v2 = [score for (term, score) in sorted(terms2.items())]

# Вычислить величину сходства документов
distances[title1][title2] = nltk.cluster.util.cosine_distance(v1, v2)

if title1 == title2:
    #print distances[title1][title2]
    continue

if distances[title1][title2] < min_dist:
    (min_dist, most_similar) = (distances[title1][title2], title2)

print(u'Most similar (score: {})\\n{}\\n{}\\n'.format(1-min_dist, title1,
                                                    most_similar))

```

Если это объяснение косинусного сходства показалось вам интересным, тогда еще более интересным для вас будет узнать, что *для обработки запросов в векторном пространстве можно использовать тот же прием вычисления сходства документов, только вместо векторов документов в сравнении должны участвовать вектор запроса и векторы документов*. Сделайте паузу и поразмышляйте над этим: это утверждение имеет надежный математический фундамент.

С точки зрения реализации программы для вычисления сходства по всему корпусу такой упрощенный подход предполагает создание вектора для запроса и его сравнение с каждым документом в корпусе. Очевидно, что прямое сравнение вектора запроса с каждым возможным вектором документа — не самая лучшая идея, даже для небольшого корпуса, поэтому вам потребуется принять некоторые инженерные решения, связанные с использованием индексов, чтобы добиться хорошей масштабируемости.

В главе 4 мы кратко затронули фундаментальную проблему снижения размерности как необходимую основу для кластеризации, и здесь мы вновь сталки-

ваемся с этой идеей. *Всякий раз, оказываясь перед задачей вычисления оценки сходства, вы почти неизбежно будете сталкиваться с необходимостью уменьшения размерности, чтобы сделать вычисления практически осуществимыми.*

Визуализация сходства документов в виде матричной диаграммы

Подход к визуализации сходства документов, представленный в этом разделе, основан на использовании графоподобных структур, где связи между документами кодируют меру сходства между ними. Это дает нам отличную возможность познакомиться с дополнительными приемами визуализации данных, реализованными в популярной библиотеке `matplotlib`, для создания высококачественных рисунков на Python. Работая с примерами кода в Jupyter Notebook, получаемые рисунки можно отображать непосредственно в блокноте с помощью объявления `%matplotlib inline`.

Код в примере 5.7 создает матрицу сходства документов, изображенную на рис. 5.4. В каждой ячейке матрицы (i, j) сохраняется разность между 1,0 и косинусным сходством между документами i и j . Массив `distances` уже был вычислен в примере 5.6. Пример кода содержит «волшебную команду» `%matplotlib` с параметром `inline`. Эта строка кода имеет смысл только в среде Jupyter Notebook, где она обеспечивает вывод изображения непосредственно между ячейками.

Пример 5.7. Создание рисунка, отражающего величину косинусного сходства между документами

```
import numpy as np
import matplotlib.pyplot as plt # pip install matplotlib
%matplotlib inline

max_articles = 15

# Получить название - ключи в словаре 'distances'
keys = list(distances.keys())

# Извлечь названия статей
titles = [l[:40].replace('\n', ' ')+...' for l in list(distances.keys())]

n_articles = len(titles) if len(titles) < max_articles else max_articles

# Создать матрицу соответствующего размера для хранения оценок сходства
similarity_matrix = np.zeros((n_articles, n_articles))

# Цикл по ячейкам в матрице
```

```

for i in range(n_articles):
    for j in range(n_articles):
        # Извлечь косинусное расстояние между статьями i и j
        d = distances[keys[i]][keys[j]]

        # Записать величину 'сходства' между статьями i и j,
        # которая определяется как 1.0 - расстояние
        similarity_matrix[i, j] = 1.0 - d

# Создать диаграмму и оси
fig = plt.figure(figsize=(8,8), dpi=300)
ax = fig.add_subplot(111)

# Отобразить значения в ячейках матрицы как цветные квадраты,
# отражающие сходство
ax.matshow(similarity_matrix, cmap='Greys', vmin = 0.0, vmax = 0.2)

# Вывести обычные метки на осях, соответствующие статьям в коллекции
ax.set_xticks(range(n_articles))
ax.set_yticks(range(n_articles))

# Вывести подписи - названия статей
ax.set_xticklabels(titles)
ax.set_yticklabels(titles)

# Повернуть на 90 градусов подписи на оси X
plt.xticks(rotation=90);

```

Этот пример нарисует матричную диаграмму, изображенную на рис. 5.4 (текстовые метки на рисунке были немного сокращены). Черные квадраты на диагонали соответствуют максимальному сходству документов с самими собой.

5.4.4. Анализ биграмм на естественном языке

Как упоминалось выше, при обработке неструктурированного текста часто упускается из виду одна проблема — огромный объем информации, которую можно получить, одновременно рассматривая несколько лексем, потому что многие понятия мы выражаем фразами, а не отдельными словами. Например, представьте, что кто-то сказал вам, что в статье наиболее часто используются слова «open» (открытый), «source» (источник) и «government» (правительство). Сможете ли вы с уверенностью сказать, что статья обсуждает «open source» (открытое программное обеспечение), «open government» (открытое правительство), и то и другое или ни то ни другое? Заранее зная автора или содержимое статьи, вы, наверное, смогли бы правильно ответить на вопрос. Но если вы полностью



Рис. 5.4. Матричная диаграмма, отображающая связи, извлеченные из текстового содержимого

полагаетесь на машинные методы, пытаться *классифицировать* документ как описывающий совместную разработку программного обеспечения или качественно новое правительство, вам потребуется вернуться к тексту и определить, какое из двух других слов чаще всего встречается после слова «ореп», то есть вам нужно найти *словосочетания*, начинающиеся со слова «ореп».

Как мы узнали в главе 4, *n*-граммы — это всего лишь краткий способ представления всех возможных последовательностей из *n* лексем в тексте и основная

структура данных для определения словосочетаний. Для любого значения n всегда существует $n - 1$ n -грамм. Например, если выбрать все биграммы (2-граммы) из последовательности ["Mr.", "Green", "killed", "Colonel", "Mustard"], мы получим четыре возможных варианта: [("Mr.", "Green"), ("Green", "killed"), ("killed", "Colonel"), ("Colonel", "Mustard")]. Для определения словосочетаний потребуется больший фрагмент текста, чем наше простое предложение, но если допустить наличие специальных знаний или дополнительного текста, тогда следующим шагом мог бы стать статистический анализ биграмм, чтобы определить, какие из них, скорее всего, являются словосочетаниями.

ТРЕБОВАНИЯ К ПАМЯТИ ДЛЯ ХРАНЕНИЯ N-ГРАММ

Стоит отметить, что для хранения n -грамм требуется столько же памяти, сколько необходимо для $(T - 1) \times n$ лексем (то есть почти $T \times n$), где T — число лексем в тексте, а n — размер n -граммы в лексемах. Например, представьте, что у нас есть документ, содержащий 1000 лексем, занимающий около 8 Кбайт памяти. Тогда для хранения всех биграмм, извлеченных из текста, потребуется примерно в два раза больше памяти, чем для исходного текста, то есть около 16 Кбайт, так как нужно будет хранить 999×2 лексем, плюс накладные расходы. Для хранения всех триграмм (998×3 лексем плюс накладные расходы) потребуется примерно в три раза больше памяти, чем для исходного текста, или 24 Кбайт. Таким образом, для хранения всех n -грамм без применения специализированных структур данных или алгоритмов сжатия понадобится примерно в n раз больше памяти, чем для исходного текста.

n -граммы дают простой и очень эффективный метод кластеризации слов, часто встречающихся вместе. Если определить все n -граммы, даже для небольшого значения n , можно без дополнительной работы обнаружить в тексте некоторые интересные закономерности. (В практике анализа данных часто используются биграммы и триграммы.) Например, анализируя биграммы, извлеченные из достаточно длинного текста, можно обнаружить собственные имена, такие как «Mr. Green» и «Colonel Mustard», понятия, такие как «open source» или «open government», и т. д. Фактически, выделение биграмм дает практически те же результаты, что и функция `collocations`, которую мы использовали ранее, за исключением дополнительного статистического анализа, учитывающего использование редких слов. Аналогичные закономерности наблюдаются при анализе часто встречающихся триграмм и n -грамм для значений n немного более 3. Как уже было показано в примере 5.4, пакет NLTK берет на себя большую часть работы по извлечению n -грамм, выявлению словосочетаний в тексте, определению контекста, в котором использована лексема, и многое другое. Пример 5.8 демонстрирует порядок определения словосочетаний.

Пример 5.8. Использование NLTK для выделения биграмм и словосочетаний из предложения

```
import nltk

sentence = "Mr. Green killed Colonel Mustard in the study with the " + \
    "candlestick. Mr. Green is not a very nice fellow."

print([bg for bg in nltk.ngrams(sentence.split(), 2)])
txt = nltk.Text(sentence.split())

txt.collocations()
```

Самый большой недостаток встроенных «демонстрационных» функций, таких как `nltk.Text.collocations`, состоит в том, что они обычно не возвращают структуры данных, которые можно хранить и обрабатывать. Всякий раз, столкнувшись с такой ситуацией, просто загляните в исходный код, который обычно довольно легко понять и адаптировать для конкретных целей. Пример 5.9 иллюстрирует, как можно определить словосочетания и индексы соответствия в коллекции лексем и сохранить контроль над результатами.



Найти каталог на диске, где хранится исходный код пакета, обычно можно с помощью интерпретатора Python, обратившись к атрибуту `__file__`. Например, попробуйте вывести значение `nltk.__file__`, чтобы определить путь к исходному коду NLTK на диске. В IPython или Jupyter Notebook для этой цели можно использовать «магическую комбинацию из двух вопросительных знаков», например, выполнив команду `nltk??`.

Пример 5.9. Использование NLTK для выделения словосочетаний подобно тому, как это делает `nltk.Text.collocations`

```
import json
import nltk
from nltk.metrics import association

# Загрузить текстовые данные на естественном языке
DATA = 'resources/ch05-textfiles/ch05-timoreilly.json'
data = json.loads(open(DATA).read())

# Число искомых словосочетаний
N = 25

all_tokens = [token for post in data for token in post['content'].lower().split()]

finder = nltk.BigramCollocationFinder.from_words(all_tokens)
```



```
finder.apply_freq_filter(2)
finder.apply_word_filter(lambda w: w in nltk.corpus.stopwords.words('english'))
scorer = association.BigramAssocMeasures.jaccard
collocations = finder.nbest(scorer, N)

for collocation in collocations:
    c = ' '.join(collocation)
    print(c)
```

Этот пример близко повторяет реализацию демонстрационной функции `collocations` из пакета NLTK. Он отбрасывает биграммы, появляющиеся реже минимального числа раз (в данном случае 2), а затем применяет метрику оценки для ранжирования результатов. В данном случае на роль такой метрики выбрано известное сходство Жаккара, обсуждавшееся в главе 4 и реализованное в виде функции `nltk.metrics.association.BigramAssocMeasures.jaccard`. Для ранжирования совместного появления слов в любой данной биграмме класс `BigramAssocMeasures` использует *таблицу сопряженности*. Концептуально метрика сходства Жаккара оценивает сходство множеств, роль которых в данном случае играют конкретные биграммы, присутствующие в тексте.

Даже притом, что это довольно сложная тема, в следующем разделе, «Таблицы сопряженности и функции оценки», мы подробно рассмотрим порядок вычисления таблиц сопряженности и значений метрики Жаккара, потому что это совершенно необходимо для более глубокого понимания алгоритма определения словосочетаний.

А пока давайте исследуем некоторые результаты, извлеченные из сообщений Тима О'Рейли (Tim O'Reilly), наглядно демонстрирующие, насколько больше информации несет ранжированный список биграмм в сравнении со списком лексем благодаря дополнительному контексту, проясняющему смысл слов:

```
brett goldstein
cabo pulmo
nick hanauer
wood fired
yuval noah
child welfare
silicon valley
jennifer pahlka
barre historical
computational biologist
drm-free ebooks
mikey dickerson
saul griffith
bay mini
```

credit card
east bay
on-demand economy
white house
inca trail
italian granite
private sector
weeks ago

Вызывает удивление, как много имен собственных и распространенных фраз было извлечено из данных, если учесть, что при этом не использовалось никаких специальных эвристик или тактик. Конечно, вы могли бы сами прочитать текст и выбрать эти имена, но способность машины сделать это за вас позволяет вам сосредоточиться на более целенаправленных видах анализа.

Конечно, в результатах присутствует неизбежный шум, потому что мы пока не предприняли никаких усилий для отделения знаков препинания от лексем, но, учитывая небольшой объем выполненной работы, мы получили очень даже неплохие результаты. Возможно, сейчас самое время упомянуть, что даже при использовании достаточно мощных средств обработки естественного языка трудно устранить весь шум из результатов анализа. И тем не менее боритесь с шумом, ищите эвристики для управления им и рано или поздно вы подойдете к точке, когда сможете получить идеальные результаты, подобные тем, которые мог бы извлечь хорошо образованный человек.

Самое главное наблюдение, которое можно сделать на данный момент, заключается в том, что, приложив очень немного усилий, и за короткое время мы смогли применить другой простой метод, чтобы извлечь некоторые осмысленные результаты из текстовых данных, и *похоже, что результаты получились довольно репрезентативными, чтобы можно было говорить об их истинности*. Это дает надежду, что применение того же метода к любому другому неструктурированному тексту может дать столь же информативные результаты, на основе которых мы сумеем быстро выявить ключевые элементы, обсуждаемые в тексте. И, что не менее важно, даже притом, что полученные результаты в основном подтверждают то, что вы могли бы знать о Тиме О'Рейли (Tim O'Reilly), вы, возможно, узнали что-то новое, о чем свидетельствуют имена людей в верхней части списка. Мы с легкостью могли бы использовать метод *concordance*, регулярное выражение или даже метод *find* встроенного строкового типа в языке Python, чтобы найти посты, имеющие отношение к «Brett Goldstein» (Бретт Голдштейн), но давайте попробуем пойти другим путем и воспользуемся кодом из примера 5.5, чтобы получить оценку TF-IDF для запроса [brett, goldstein]. Вот что получилось:

Title: Brett Goldstein gives some excellent advice on basic security hygiene...
Score: 0.19612432637

В результате целевой запрос привел нас к статье с советами по безопасности. Мы фактически начали с номинального (по большей части) анализа текста, выделили некоторые интересные темы, применив анализ словосочетаний, и отыскиали в тексте одну из этих тем с помощью оценки TF-IDF. Точно так же мы могли бы использовать косинусное сходство, чтобы найти статью, похожую на что-то еще, интересующее нас.

Таблицы сопряженности и функции оценки



В этом разделе подробно обсуждаются некоторые технические детали работы функции вычисления оценки Жаккара в классе `BigramCollocationFinder`, использованной в примере 5.9. Если вы читаете эту главу впервые или вас не интересуют эти детали, можете смело пропустить этот раздел и вернуться к нему позже. Это довольно сложная тема, и ее не обязательно полностью понимать, чтобы эффективно использовать приемы, описанные в этой главе.

Для вычисления оценок, используемых при ранжировании биграмм, часто используется структура данных, которая называется *таблицей сопряженности*. Эта таблица предназначена для компактного представления частот вариантов появления разных слов в биграммах. Взгляните на элементы в табл. 5.6, выделенные жирным, где *token1* обозначает присутствие *token1* в биграмме, а *~token1* — отсутствие.

Таблица 5.6. Пример таблицы сопряженности — значения, оформленные наклонным шрифтом, представляют «границы», а значения, оформленные жирным шрифтом, — частоты вариантов биграмм

	token1	~token1	
<i>token2</i>	frequency(token1, token2)	frequency(~token1, token2)	<i>frequency(*, token2)</i>
<i>~token2</i>	frequency(token1, ~token2)	frequency(~token1, ~token2)	
	<i>frequency(token1, *)</i>		<i>frequency(*, *)</i>

Несмотря на некоторые тонкости, связанные с тем, какие ячейки имеют значение для тех или иных вычислений, нетрудно заметить, что четыре ячейки в середине таблицы выражают частоты присутствия разных лексем в биграмме.

Значения в этих ячейках можно использовать для вычисления разных метрик сходства с целью последующего ранжирования биграмм, как это было сделано в примере 5.9, где определялось сходство Жаккара, к которому мы вернемся чуть позже. Но прежде давайте кратко обсудим, как вычисляются элементы таблицы сопряженности.

Способ вычисления различных элементов таблицы сопряженности напрямую зависит от структур данных, которые были предварительно получены некоторым способом. Если предположить, что у вас имеются только частоты появления разных биграмм в тексте, тогда значение $frequency(token1, token2)$ легко определить прямым поиском, но как получить значение $frequency(\sim token1, token2)$? В отсутствие любой другой информации придется подсчитать *все биграммы*, где на втором месте находится $token2$, и вычесть $frequency(token1, token2)$ из этой суммы. (Если это объяснение вызывает у вас сомнения, найдите время и выполните расчеты вручную, чтобы убедиться в его истинности.)

Если кроме частот биграмм имеются также частоты появления отдельных лексем в тексте (униграмм), можно использовать другой, *более простой путь*, вовлекающий две операции поиска и одну арифметическую операцию. Вычитите из количества появлений униграммы $token2$ количество появлений биграммы $(token1, token2)$, и вы получите количество появлений биграммы $(\sim token1, token2)$. Например, если биграмма («*mr.*», «*green*») встречается в тексте три раза, а униграмма («*green*») — семь раз, это означает, что биграмма $(\sim \text{«mr.»}, \text{«green»})$ встречается четыре раза (где $\sim \text{«mr.»}$ буквально означает: «любая лексема, отличная от ‘mr.’»). Выражение $frequency(*, token2)$ в табл. 5.6 представляет униграмму $token2$ и называется *пограничным*, потому что находится на границе таблицы — оно помогает сократить вычисления. Значение $frequency(token1, *)$ точно так же помогает сократить вычисление $frequency(token1, \sim token2)$, а выражение $frequency(*, *)$ обозначает количество всех возможных униграмм и эквивалентно общему числу лексем в тексте. Зная $frequency(token1, token2)$, $frequency(token1, \sim token2)$ и $frequency(\sim token1, token2)$, с помощью $frequency(*, *)$ легко вычислить $frequency(\sim token1, \sim token2)$.

Обсуждение таблиц сопряженности может показаться несколько отвлекающим, тем не менее эта тема важна для понимания особенностей работы разных функций вычисления оценок. Например, рассмотрим метрику сходства Жаккара, представленную в главе 4. Концептуально она выражает сходство двух множеств и определяется как:

$$\frac{|\text{Множество } 1 \cap \text{Множество } 2|}{|\text{Множество } 1 \cup \text{Множество } 2|}.$$

Иначе говоря, это отношение количества элементов, общих для двух множеств, к общему количеству уникальных элементов в этих множествах. Стоит приостановиться ненадолго и поразмышлять над этой простой и эффективной оценкой. Если множества *Set1* и *Set2* идентичны, их объединение и пересечение должны быть эквивалентны друг другу, в результате чего отношение дает значение 1,0. Если множества полностью отличны, в числителе получится значение 0 и результатом будет значение 0,0. Все остальные варианты будут давать в результате значение между этими двумя числами.

Применительно к конкретной биграмме сходство Жаккара выражает отношение частоты появления конкретной биграммы к сумме частот биграмм, содержащих слова из данной биграммы. Эту метрику можно интерпретировать примерно так: чем больше это отношение, тем чаще появляется биграмма (*token1*, *token2*) в тексте и тем выше вероятность, что словосочетание «*token1 token2*» выражает осмысленное понятие.

Выбор подходящей функции оценки обычно зависит от знания базовых характеристик данных, некоторой интуиции, а иногда и удачи. Большинство ассоциативных метрик определяется в модуле `nlk.metrics.association`, который подробно обсуждается в главе 5 книги Кристофера Маннинга (Christopher Manning) и Хайнриха Шютце (Hinrich Schütze) *Foundations of Statistical Natural Language Processing* (MIT Press), которая доступна онлайн¹ и может служить превосходным справочником по теме, обсуждаемой далее.

ПОЧЕМУ НОРМАЛЬНОЕ РАСПРЕДЕЛЕНИЕ ТАК ВАЖНО?

Одним из фундаментальных понятий в статистике является нормальное распределение. Это распределение, известное также как *колоколообразное* из-за формы его кривой на графике, называют нормальным, потому что оно часто используется как основа (или норма), с которой сравниваются другие распределения. Это симметричное распределение является, пожалуй, самым широко используемым в статистике. Одна из причин высокой значимости нормального распределения состоит в том, что его модель изменчивости встречается во многих естественных явлениях, от физических характеристик популяций до отклонений в производственных процессах и результатах выбрасывания игральные костей.

Одним из практических правил, объясняющих важность нормального распределения, может служить так называемое *правило 68–95–99*,² — удобная эвристика, помогающая дать ответы на многие вопросы о распределениях, близких к нормальному. Как оказывается, в нормальном распределении почти все данные (99,7%) располагаются в пределах трех

¹ <http://stanford.io/1a1mBQy>.

² <http://bit.ly/1a1mEf0>.

стандартных отклонений от среднего, 95% — в пределах двух стандартных отклонений и 68% — в пределах одного стандартного отклонения. То есть если известно, что некоторое явление объясняется распределением, близким к нормальному, и определены его среднее и стандартное отклонение, эти сведения могут помочь вам ответить на многие вопросы. Суть правила 68–95–99,7 иллюстрирует рис. 5.5.

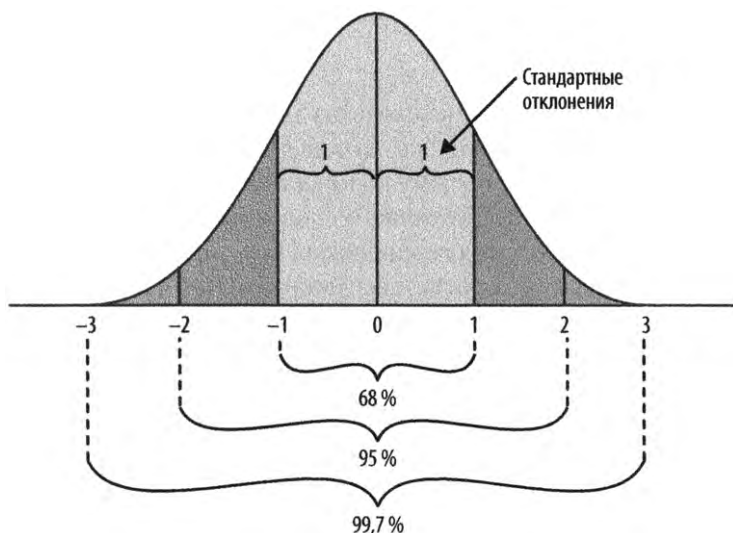


Рис. 5.5. Нормальное распределение является основой математической статистики, потому что моделирует изменчивость многих естественных явлений

На сайте «Академия Хана» можно найти замечательный 30-минутный видеобзор «Introduction to the Normal Distribution»¹ нормального распределения; вам также может понравиться 10-минутный фрагмент из обзора центральной предельной теоремы², столь же основополагающего понятия в статистике, в которой нормальное распределение проявляется неожиданным (и удивительным) образом.

Рассмотрение всех этих метрик выходит за рамки этой книги, однако подробное их описание с примерами вы найдете в главе, упомянутой выше. Если вам понадобится реализовать свой механизм определения словосочетаний, хорошими отправными точками вам послужат сходство Жаккара, коэффициент сходства Дайса (Dice) и коэффициент подобия. Они кратко описываются ниже вместе с некоторыми другими важными критериями.

¹ <http://bit.ly/1a1mCnm>.

² <http://bit.ly/1a1mCnA>.

Простая частота

Как следует из названия, простая частота — это отношение количества вхождений определенной n -граммы к количеству всех n -грамм. Эта метрика может пригодиться для анализа общей частоты появления определенного словосочетания в тексте.

Сходство Жаккара

Сходство Жаккара — это оценка сходства двух множеств. В случае со словосочетаниями она определяется как отношение количества вхождений конкретного словосочетания к общему количеству словосочетаний, содержащих хотя бы одно слово из интересующего словосочетания. Эта метрика может пригодиться для определения вероятности, что данные слова действительно образуют словосочетание, а также для ранжирования вероятных словосочетаний. Используя обозначения, согласующиеся с предыдущими объяснениями, математическую формулу вычисления этой оценки можно записать так:

$$\frac{\text{частота}(\text{словосочетание1}, \text{словосочетание2})}{\text{частота}(\text{словосочетание1}, \text{словосочетание2}) + \text{частота}(\sim \text{словосочетание1}, \text{словосочетание2}) + \text{частота}(\text{словосочетание1}, \sim \text{словосочетание2})}$$

Коэффициент сходства Дайса

Коэффициент сходства Дайса очень похож на оценку сходства Жаккара. Принципиальное отличие состоит в удвоении числителя отношения. Математически этот коэффициент определяется так:

$$\frac{2 \times \text{частота}(\text{словосочетание1}, \text{словосочетание2})}{\text{частота}(*, \text{словосочетание2}) + \text{частота}(\text{словосочетание1}, *)}$$

Легко доказать, что математически:

$$\text{Dice} = \frac{2 \times \text{сходство Жаккара}}{1 + \text{сходство Жаккара}}$$

Эта метрика часто выбирается вместо сходства Жаккара, когда желательно увеличить оценку в пользу перекрытия множеств при большом количестве различий между ними. Причина в том, что оценка Жаккара по своей природе уменьшается с увеличением различий между множествами из-за объединения множеств в знаменателе.

t-критерий Стьюдента

t-критерий Стьюдента традиционно используется для проверки гипотез. Применительно к *n*-граммам с помощью *t*-критерия можно проверить гипотезу о принадлежности двух слов одному словосочетанию. Процедура статистических вычислений использует стандартное распределение с нормированием. Преимущество *t*-критерия перед простой частотой состоит в том, что *t*-критерий учитывает частоту биграммы относительно составляющих ее компонентов. Эта характеристика упрощает ранжирование устойчивых словосочетаний. Однако она предполагает нормальное распределение словосочетаний, что не всегда верно.

Хи-квадрат

Как и *t*-критерий Стьюдента, эта метрика часто применяется для проверки независимости двух переменных и может использоваться для оценки принадлежности двух слов одному словосочетанию путем вычисления критерия Пирсона хи-квадрат, выражающего статистическую значимость. Вообще говоря, различия в результатах, полученных применением *t*-критерия и критерия хи-квадрат, часто оказываются несущественными. Но в отличие от *t*-критерия, критерий хи-квадрат не предполагает нормального распределения словосочетаний, поэтому он чаще используется в практике.

Коэффициент подобия

Эта метрика представляет еще один подход к проверке гипотез и используется для измерения независимости между словами, которые могут образовывать словосочетание. Практические исследования показали, что в общем случае она лучше подходит для определения словосочетаний, чем критерий хи-квадрат, и дает хорошие результаты для текстов, включающих множество редких словосочетаний. Конкретная реализация вычислений оценок вероятности для словосочетаний в NLTK предполагает биномиальное распределение¹, где параметры распределения вычисляются по количеству вхождений словосочетаний и их членов.

Поточечная взаимная информация

Поточечная взаимная информация (pointwise mutual information, PMI) — это мера количества информации, получаемой о конкретном слове, если известно значение соседнего слова. То есть эта оценка сообщает о том, как

¹ <http://bit.ly/1a1mEMj>.

много одно слово может рассказать о другом. По иронии (в контексте текущего обсуждения) метрика PMI оценивает часто встречающиеся слова ниже, чем редкие, что противоречит обсуждаемой цели. Поэтому данная метрика может служить хорошей оценкой независимости, но не подходит для оценки зависимости (то есть это далеко не идеальный выбор для оценки словосочетаний). Также было показано, что оценка PMI непригодна для применения к разреженным данным, и другие методы, такие как коэффициент подобия, обычно превосходят ее.

Оценка и выбор лучшего метода для применения в конкретной ситуации — это часто больше искусство, чем наука. Некоторые задачи хорошо изучены и могут использоваться как основа для дополнительных изысканий, тогда как в других обстоятельствах требуется проведение полноценных исследований и экспериментов. Приступая к решению нетривиальной задачи, вы можете попробовать определить, насколько хорошо она изучена, ознакомившись с научной литературой (книгами или академическими статьями, которые можно найти на сайте Google Scholar¹).

5.4.5. Размышления об анализе данных на естественном языке

В этой главе были представлены разные инструменты и приемы анализа данных на естественном языке, и некоторые заключительные размышления могут помочь вам обобщить ее содержание.

Контекст определяет значение

TF-IDF — мощный и простой в использовании инструмент, но наша конкретная реализация имеет несколько существенных ограничений, о которых мы умолчали. Наиболее важное из них — данная оценка интерпретирует документ как «мешок слов», то есть она не учитывает порядок слов в документе и в запросе. Например, для запроса «Green Mt.» она вернет тот же результат, что и для запроса «Mt. Green», если не реализовать логику, принимающую в расчет порядок слов в запросе или интерпретирующую запрос как фразу, а не как набор независимых слов. Как мы уже знаем, порядок следования слов играет важную роль.

Применяя анализ n -грамм для учета словосочетаний и порядка следования слов, мы все еще сталкиваемся с проблемой, когда TF-IDF приписывает

¹ <http://bit.ly/1a1mHYk>.

одинаковый смысл всем одинаковым лексемам. Однако в действительности это далеко не так. Омонимы¹ — это слова, одинаковые по написанию и произношению, но разные по значению, смысл которых определяется контекстом. Такие примеры омонимов, как *коса*, *ключ*, *дрось* и *горн*, наглядно показывают важность контекста для определения смысла слов.

Косинусное сходство страдает практически теми же недостатками, что и TF-IDF. Оно не учитывает ни контекст документа, ни порядок слов в n -граммах и предполагает сходство слов, находящихся близко друг к другу в векторном пространстве, что, конечно же, не всегда верно. Как и в случае с оценкой TF-IDF, очевидным контрпримером являются омонимы. Наша реализация вычисления косинусного сходства для оценки относительной важности слов в документах также зависит от TF-IDF, из-за чего ей свойственны все недостатки TF-IDF.

Естественный язык перегружен контекстом

Возможно, вы заметили, что анализ неструктурированного текста сопряжен с множеством досадных мелочей, и эти мелочи оказываются важными для конкурирующих реализаций. Например, сравнение строк выполняется с учетом регистра символов, поэтому важно нормализовать текст так, чтобы частоты слов вычислялись с максимальной точностью. Простая нормализация, выражающаяся в приведении всех символов к нижнему регистру, может также усложнить ситуацию, поскольку регистр символов в некоторых словах может иметь важное значение.

«Mr. Green» и «Web 2.0» — вот пара особенно показательных примеров. Если в слове «Green» из словосочетания «Mr. Green» оставить первую букву заглавной, это дало бы алгоритму обработки запроса полезную подсказку, что данное слово, скорее всего, является не прилагательным, а частью именной конструкции. Мы еще вернемся к этой теме, когда будем обсуждать приемы обработки естественного языка в главе 6, потому что в модели «мешок слов» теряется *контекст*, в котором используется слово «Green», тогда как продвинутые методы обработки естественного языка сохраняют его.

Анализ контекста естественного языка — сложная задача

Еще одна проблема в нашей реализации, оказывающая еще большее влияние, чем особенности TF-IDF: функция `split`, используемая для разбиения

¹ <http://bit.ly/1a1mFzJ>.

текста на лексемы, оставляет знаки препинания в составе лексем, что сказывается на точности подсчета частот. Например, `corpus[ 'b' ]` в примере 5.2 завершается лексемой «study.», которая считается отличной от лексемы «study» в `corpus[ 'a' ]` (и которая с большей вероятностью будет найдена в запросе). В данном случае завершающая точка в лексеме влияет на точность вычислений TF и IDF. Такие простые знаки препинания, как точка, сигнализирующая о конце предложения, образуют контекст, который легко распознается нашим мозгом, но компьютеру намного сложнее добиться такого же уровня точности.

Разработка программного обеспечения, помогающего компьютерам лучше понимать контекст слов в естественном языке, является очень активной областью исследований и имеет огромный потенциал для технологий поиска, интернета и искусственного интеллекта.

5.5. Заключительные замечания

В этой главе мы познакомились с некоторыми способами очистки данных на естественном языке и основами теории информационного поиска, такими как TF-IDF, косинусное сходство и анализ словосочетаний в собираемых нами данных. Наконец, мы дошли до того, что рассмотрели несколько проблем, которые приходится решать любому разработчику поисковых систем, чтобы создать успешный технологический продукт. Я надеюсь, что эта глава помогла вам получить хорошее представление о том, как извлечь полезную информацию из неструктурированного текста, но она лишь едва коснулась самых важных теоретических понятий и технических аспектов. Информационный поиск — это огромная индустрия с многомиллиардным оборотом, поэтому вы легко можете представить сумму инвестиций, которые вкладываются в теоретические исследования и практические реализации владельцами таких поисковых систем, как Google и Bing.

Учитывая огромную мощь поисковых систем, таких как Google, легко забыть даже о существовании этих фундаментальных методов поиска. Однако их понимание дает представление о текущих допущениях и ограничениях в поиске, а также позволяет четко отличать новейшие методы поиска сущностей. Глава 6 представляет качественно иную парадигму с уходом от традиционной прак-

тики использования приемов, рассмотренных в этой главе. Технологические компании обладают множеством захватывающих возможностей эффективного анализа данных на естественном языке.



Исходный код примеров для этой и всех других глав доступен на GitHub¹ в удобном формате Jupyter Notebook, который вы можете опробовать, не покидая веб-браузера.

5.6. Упражнения

- Воспользуйтесь возможностями построения графиков в Jupyter Notebook, о которых рассказывалось в главе 1, и отобразите график кривой Ципфа для лексем в корпусе.
- Если у вас появится желание применить приемы, описанные в этой главе, к данным из интернета, попробуйте воспользоваться простым в применении и мощным фреймворком Scrapy, который поможет вам выполнить обход веб-страниц и собрать текстовые данные.
- Найдите время и добавьте интерактивные возможности в сценарий построения матричной диаграммы, представленный в этой главе. Попробуйте, например, добавить обработчик событий, автоматизирующий переход к сообщению в случае щелчка на тексте. Придумайте другие, не лишённые смысла способы упорядочения строк и столбцов, чтобы упростить выявление закономерностей.
- Измените код, формирующий данные в формате JSON для матричной диаграммы, так, чтобы он вычислял оценку сходства иначе и, соответственно, по-другому оценивал близость документов.
- Какие, по вашему мнению, дополнительные признаки в тексте могли бы сделать оценку сходства документов более точной?
- Найдите время и попробуйте глубже изучить теоретические основы информационного поиска, представленные в этой главе.

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

5.7. Онлайн-ресурсы

Ниже приводится список ссылок, упоминавшихся в этой главе, которые могут оказаться полезными для вас:

- Правило 68–95–99,7¹.
- Биномиальное распределение².
- Корпус Brown Corpus³.
- Центральная предельная теорема⁴.
- Обзор HTTP API⁵.
- Книга *Introduction to Information Retrieval*⁶.
- Видеообзор «Introduction to the Normal Distribution»⁷.
- Глава 5 («Collocations») в книге *Foundations of Statistical Natural Language Processing*⁸.
- `matplotlib`⁹.
- Онлайн-книга NLTK¹⁰.
- Фреймворк Scrapy¹¹.
- Закон Ципфа¹².

¹ <http://bit.ly/1a1mEf0>.

² <http://bit.ly/1a1mEMj>.

³ <http://bit.ly/1a1mB2X>.

⁴ <http://bit.ly/1a1mCnA>.

⁵ <http://bit.ly/1a1mAfm>.

⁶ <http://stanford.io/1a1mAvP> (Кристофер Д. Маннинг, Прабхакар Рагхаван, Хайнрих Шютце. Введение в информационный поиск. М.: Вильямс, 2014. — *Примеч. пер.*)

⁷ <http://bit.ly/1a1mCnm>.

⁸ <http://stanford.io/1a1mBQy>.

⁹ <https://matplotlib.org/>.

¹⁰ <http://bit.ly/1a1mtAk>.

¹¹ <http://bit.ly/1a1mG6P>.

¹² <http://bit.ly/1a1mCUD>.

6

Анализ веб-страниц: использование методов обработки естественного языка, обобщение статей из блогов и многое другое

Эта глава продолжает обсуждение, начатое в пятой главе, и является скромной попыткой познакомить вас с приемами обработки естественного языка (natural language processing, NLP) и особенностями их применения к обширному источнику данных на человеческом языке¹, с которым вы столкнетесь в социальных сетях (или где-то еще). В предыдущей главе мы познакомились с некоторыми основополагающими методами из теории информационного поиска (information retrieval, IR), обычно интерпретирующими текстовые документы как «мешки слов» (неупорядоченные коллекции слов), которые можно представлять в виде векторов и манипулировать ими как векторами. Даже притом, что эти модели позволяют получить неплохие результаты во многих случаях, они страдают существенным недостатком — не учитывают контекст, от которого зависят значения слов.

В этой главе демонстрируются различные методы, которые в большей мере зависят от контекста и глубже погружаются в семантику данных на человеческом языке. Конечно, веб-API социальных сетей, возвращающие данные со строго определенной структурой, играют важную роль, но самой главной валютой человеческого общения являются данные на естественном языке, такие как слова, которые вы читаете на этой странице, в постах в Facebook, в веб-страницах, на которые ссылаются твиты, и т. д. Данные на человеческом языке — самый распространенный вид доступных нам данных, и будущее инноваций в обработке данных в значительной мере зависит от нашей спо-

¹ На протяжении всей этой главы фраза *данные на человеческом языке* будет подразумевать объект методов обработки естественного языка и обозначать то же самое, что и фразы *данные на естественном языке* или *неструктурированные данные*. Выбор этих слов не предполагает какого-то особого значения, кроме точности описания самих данных.

способности эффективно использовать компьютеры для понимания цифровых форм человеческого общения.



Настоятельно рекомендуем вам разобраться в содержимом предыдущей главы, прежде чем продолжить читать эту главу. Для понимания NLP необходимо знать и понимать основные сильные и слабые стороны TF-IDF, моделей векторных пространств и т. д. По этой причине эта и предыдущая главы связаны друг с другом намного теснее, чем любые другие главы в этой книге.

По аналогии с предыдущими главами постараемся охватить минимальное количество деталей, чтобы вы могли получить общее представление об этой довольно сложной теме, а также дадим достаточный объем технической информации, чтобы вы могли приступить к самостоятельному анализу данных. Срезая углы, чтобы дать вам важнейшие 20% навыков, которые вы сможете использовать для выполнения 80% работы (никакая глава ни в какой книге — или даже небольшое многотомное издание, если уж на то пошло, — не смогут осветить тему NLP в полном объеме), мы постарались организовать эту главу как практическое введение, содержащее ровно столько сведений, сколько необходимо, чтобы можно было начать исследовать данные на человеческом языке, которые вы найдете повсюду в социальных сетях. Несмотря на то что мы сосредоточились на извлечении данных на человеческом языке из веб-страниц, имейте в виду, что почти каждый веб-сайт со своим API будет возвращать такие данные, поэтому описываемые здесь методы в равной степени применимы практически к любому социальному веб-сайту.



Загрузите последний исправленный исходный код примеров для этой (и любой другой) главы, доступный по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Воспользуйтесь также преимуществами виртуальной машины, описанными в приложении А, чтобы получить максимальное удовольствие от опробования примеров кода.

6.1. Обзор

Эта глава продолжает наше путешествие по методам анализа данных на человеческом языке и использует в качестве основы произвольные веб-страницы и ленты. В этой главе вы узнаете, как:

- загружать веб-страницы и извлекать из них данные на человеческом языке;
- использовать NLTK для решения основных задач в обработке естественного языка;

- выполнить контекстно-зависимый анализ в NLP;
- использовать NLP для решения таких аналитических задач, как создание краткого изложения документов;
- оценить качество в областях, связанных с прогнозным анализом.

6.2. Скрапинг, парсинг и обход сайтов в интернете

Извлечение веб-страниц из интернета не представляет большой сложности, если воспользоваться каким-либо языком программирования или утилитой командной строки, такой как `curl` или `wget`, но извлечение содержательного текста из страницы — задача весьма нетривиальная. Конечно, страница содержит нужный нам текст, но она также содержит значительный объем другого «шаблонного» контента, такого как панели навигации, заголовки, колонтитулы, рекламные объявления и другие источники шума, которые в большинстве случаев не представляют особого интереса. То есть проблема в том, что простого удаления HTML-тегов недостаточно, потому что шаблонное содержимое после удаления тегов никуда не исчезает. Хуже того, иногда на странице может быть больше шаблонного содержимого, создающего шум, чем полезной информации.

Но не впадайте в уныние, потому что на протяжении многих лет продолжали совершенствоваться инструменты, помогающие определить интересующее содержимое, и сейчас в вашем распоряжении есть несколько отличных вариантов, помогающих выделить содержательную информацию для последующего анализа. Кроме того, относительно широкая распространенность лент новостей, таких как RSS и Atom, часто может помочь процессу извлечения чистого текста без посторонних примесей, так характерных для веб-страниц, если вас заинтересует возможность использовать ленты новостей для извлечения текста, пока они существуют.



Часто в лентах новостей публикуется только свежее содержимое, поэтому иногда вам может понадобиться обрабатывать веб-страницы, даже если ленты новостей будут доступными. Если у вас есть выбор, возможно, вы предпочтете ленты новостей, но вообще вы должны быть готовы извлекать из обоих источников.

Одним из замечательных инструментов для *веб-скрапинга* (так называется процесс извлечения текста из веб-страниц) является библиотека `boilerpipe`¹

¹ <http://bit.ly/2MzPXhy>.

для Java, которая умеет выявлять и удалять шаблонное содержимое из веб-страниц. Библиотека основана на теории, рассматриваемой в статье под названием «Boilerplate Detection Using Shallow Text Features»¹, где рассказывается об эффективном использовании методов машинного обучения с учителем² для отделения шаблонного и содержательного содержимого страницы. Приемы обучения с учителем включают процесс создания прогнозной модели на обучающей выборке, репрезентативной для своей предметной области, поэтому вы сможете настроить `boilerpipe` и добиться более высокой точности.

В библиотеке имеется универсальный экстрактор, используемый по умолчанию; экстрактор, обученный для извлечения текста из веб-страниц со статьями; и экстрактор, обученный извлекать самый длинный текст, имеющийся на странице, который с успехом можно применять к веб-страницам, содержащим только один большой блок текста. В любом случае текст может потребовать дополнительной обработки после извлечения, в зависимости от того, какие другие признаки используются для идентификации шума или полезной информации, но основную работу за вас выполнит `boilerpipe`.

Библиотека написана на Java, но она получила настолько широкую популярность, что для нее был написан пакет-обертка³ для Python 3. Установить этот пакет можно командой `pip install boilerpipe3`. Наличие в системе достаточно свежей версии Java — это все, что нужно, чтобы использовать `boilerpipe`.

В примере 6.1 приводится сценарий, иллюстрирующий простоту использования этой библиотеки для решения задачи извлечения основного содержимого статьи, в соответствии с аргументом `ArticleExtractor`, который передается в конструктор `Extractor`. Также есть возможность опробовать `boilerpipe` онлайн⁴, чтобы увидеть разницу между этим и некоторыми другими экстракторами, такими как `LargestContentExtractor` или `DefaultExtractor`.

Пример 6.1. Использование `boilerpipe` для извлечения текста из веб-страницы

```
from boilerpipe.extract import Extractor

URL='http://radar.oreilly.com/2010/07/louvre-industrial-age-henry-ford.html'

extractor = Extractor(extractor='ArticleExtractor', url=URL)

print(extractor.getText())
```

¹ <http://bit.ly/1a1mN21>.

² <http://bit.ly/1a1mPHr>.

³ <http://bit.ly/2sCIFET>.

⁴ <http://bit.ly/1a1mSTF>.

Раньше веб-скрапинг был единственным способом извлечения содержимого из веб-страниц, но теперь имеется потенциально более простой путь сбора контента — из лент новостей, блогов и других синдицируемых источников. Но прежде давайте немного погрузимся в прошлое.

Пользующиеся Всемирной паутиной достаточно давно могут вспомнить время в конце 1990-х, когда инструментов для чтения новостей еще не существовало. В те времена, если вы хотели узнать, что нового появилось на веб-сайте, вы должны были просто зайти на этот сайт и проверить наличие изменений. Для устранения этого неудобства были разработаны форматы синдикации, использующие механизмы публикации в блогах, и форматы, такие как RSS (really simple syndication — действительно простое синдицирование) и Atom, основанные на развивающихся спецификациях XML¹, быстро ставшие популярными у поставщиков и потребителей контента, публикующих его и подписывающихся на него. Ленты новостей проще поддаются парсингу, потому что имеют правильно сформированный² код XML, соответствующий³ опубликованной схеме, в то время как веб-страницы могут быть оформлены с нарушениями стандартов и рекомендаций.

Для парсинга лент новостей часто используется пакет `feedparser` для Python — достойный инструмент, чтобы иметь его в своем арсенале. Установить его можно с помощью диспетчера `pip`, выполнив команду `pip install feedparser` в терминале. Пример 6.2 иллюстрирует простейший способ использования этого пакета для извлечения текста новости, ее названия и адреса URL источника из записи в ленте RSS.

Пример 6.2. Использование `feedparser` для извлечения текста (и других полей) из ленты RSS или Atom

```
import feedparser

FEED_URL='http://feeds.feedburner.com/oreilly/radar/atom'

fp = feedparser.parse(FEED_URL)

for e in fp.entries:
    print(e.title)
    print(e.links[0].href)
    print(e.content[0].value)
```

¹ <http://bit.ly/18RFKaW>.

² <http://bit.ly/1a1mQLr>.

³ <http://bit.ly/1a1mTqE>.

HTML, XML и XHTML

Еще в начале развития Всемирной паутины была подмечена сложность отделения содержимого страницы от представления, и в качестве решения (частичного) было предложено использовать XML. Идея состояла в том, чтобы создатели контента публиковали данные в формате XML и использовали таблицы стилей для их преобразования в разметку XHTML перед отображением на стороне конечного пользователя. XHTML — это по сути тот же формат HTML, но оформленный так же строго, как XML: каждый тег записывается символами нижнего регистра, теги должны правильно вкладываться друг в друга, чтобы образовать надлежащую древовидную структуру, и теги должны быть самозакрывающимися как `<br />`, или каждому открывающему тегу (например, `<p>`) обязательно должен соответствовать закрывающий тег (`</p>`).

В контексте веб-скрапинга эти соглашения обещали дополнительные преимущества в виде упрощения парсинга веб-страниц, и в контексте оформления казалось, что XHTML — это именно то, что нужно Всемирной паутине. Это предложение давало многое и почти ничего не теряло: правильно оформленное содержимое в формате XHTML можно проверить на соответствие схеме XML и пользоваться всеми другими преимуществами XML, такими как использование пространств имен с нестандартными атрибутами (прием, на который опираются технологии семантической Всемирной паутины, такие как RDFa).

Проблема в том, что это соглашение не прижилось. И теперь мы живем в мире, где продолжает процветать семантическая разметка, основанная на стандарте HTML 4.01, которому уже больше десяти лет, а XHTML и технологии, основанные на этом формате, такие как RDFa, почти не находят применения. (Фактически, такие библиотеки, как BeautifulSoup¹, создавались с конкретной целью — дать возможность обрабатывать код HTML, который оформлен с ошибками или вообще не является действительной разметкой.) Большинство веб-разработчиков затаив дыхание надеется, что HTML5² действительно даст давно назревшее решение, необходимое для использования технологии микроданных³ и дальнейшего развития инструментов публикации. Если вам интересна эта история, прочитайте статью в «Википедии»⁴.

Обход сайтов в интернете логически продолжает идеи, представленные в этом разделе: обычно он заключается в извлечении гиперссылок из страницы с последующим систематическим обходом страниц по ссылкам и извлечением их содержимого. Этот процесс многократно повторяется на произвольную глубину, в зависимости от ваших целей. Именно так работали ранние поисковые системы и продолжают работать большинство современных поисковых систем, которые индексируют содержимое Всемирной паутины. Несмотря на то что обход веб-страниц выходит за рамки нашего обсуждения, вам будет полезно

¹ <http://bit.ly/1a1mRit>.

² <http://bit.ly/1a1mRz5>.

³ <http://bit.ly/1a1mRPA>.

⁴ <http://bit.ly/1a1mS66>.

иметь некоторые практические познания, поэтому давайте отвлекемся ненадолго и поразмышляем о сложностях, связанных со сбором всех таких страниц.

Для реализации своего веб-робота, выполняющего обход веб-страниц, если вам это когда-нибудь понадобится, можно использовать замечательный фреймворк Scrapy¹ на языке Python. Для фреймворка имеется превосходная документация, включающая все необходимое, чтобы можно было с минимальными усилиями реализовать своего веб-робота. Следующий раздел немного отклоняется от основной темы и кратко описывает сложности реализации веб-роботов, чтобы вы лучше представляли, с чем можно столкнуться, пойдя по этому пути.



В настоящее время есть возможность получить периодически обновляемые корпуса веб-страниц, пригодные для большинства исследований, из таких источников, как Amazon Common Crawl², который содержит более 5 миллиардов веб-страниц с общим объемом более 81 Тбайт!

6.2.1. Обход страниц методом поиска в ширину



Этот раздел содержит некоторую информацию и анализ, касающиеся реализации веб-роботов, и не имеет прямого отношения к теме, обсуждаемой в этой главе (хотя многие из вас найдут эту информацию любопытной и поучительной). Если вы читаете эту главу впервые, можете смело пропустить этот раздел и вернуться к нему позже.

Основной алгоритм обхода веб-страниц — *поиск в ширину*³ — является фундаментальным методом исследования пространства, которое обычно моделируется как дерево или граф с заданным начальным узлом и в отсутствие любой другой информации, кроме набора возможностей. В нашем случае начальным узлом является заданная веб-страница, а соседними узлами — другие страницы, на которые имеются гиперссылки.

Существуют также альтернативные алгоритмы поиска, включая *поиск в глубину*⁴. Выбор конкретного алгоритма часто зависит от доступных вычислительных ресурсов, знаний предметной области и даже от теоретических соображений. Поиск в ширину — достаточно разумный выбор при исследовании фрагмента

¹ <http://bit.ly/1a1mG6P>.

² <http://amzn.to/1a1mXXb>.

³ <http://bit.ly/1a1mYdG>.

⁴ <http://bit.ly/1a1mVPd>.

Всемирной паутины. В примере 6.3 представлен псевдокод, иллюстрирующий работу этого алгоритма.

Пример 6.3. Псевдокод, реализующий поиск в ширину

Создать пустой граф

Создать пустую очередь для запоминания узлов, подлежащих обработке

Добавить в граф начальную точку как корневой узел

Добавить корневой узел в очередь для обработки

Повторять до достижения максимальной глубины или пока не опустеет очередь:

 Удалить узел из очереди

 Для каждого соседа этого узла:

 Если соседний узел еще не обработан:

 Добавить в очередь

 Добавить в граф

 Создать ребро в графе, связывающее узел с его соседом

Не будем тратить много времени на анализ этого подхода, потому что поиск в ширину — фундаментальный алгоритм, который должен иметь каждый в своем арсенале. В общем случае алгоритм всегда должен оцениваться по двум критериям: эффективность и результативность (или, другими словами, производительность и качество).

Стандартный анализ работы любого алгоритма обычно заключается в оценке временной и пространственной сложности для худшего случая — иначе говоря, в оценке времени, необходимого программе для выполнения, и объема памяти, требуемого для обработки очень большого набора данных. Поиск в ширину, который мы выбрали для реализации веб-робота, на самом деле не является поиском, потому что мы не ищем ничего конкретного, а просто продолжаем обход страниц до достижения заданной глубины или до исчерпания узлов в очереди. Если бы мы искали что-то определенное, а не просто перебирали ссылки до бесконечности, тогда такой подход можно было бы назвать поиском в ширину. Такой обобщенный вариант поиска в ширину называется *ограниченным поиском в ширину* и накладывает ограничение на максимальную глубину поиска, как в этом примере.

Временную и пространственную сложность поиска в ширину (или, точнее, обхода в ширину) для худшего случая можно ограничить величиной b^d , где b — коэффициент ветвления графа, а d — глубина. Если изобразить наш пример на бумаге, как показано на рис. 6.1, и поразмышлять над ним, тогда суть оценки сложности станет понятнее.

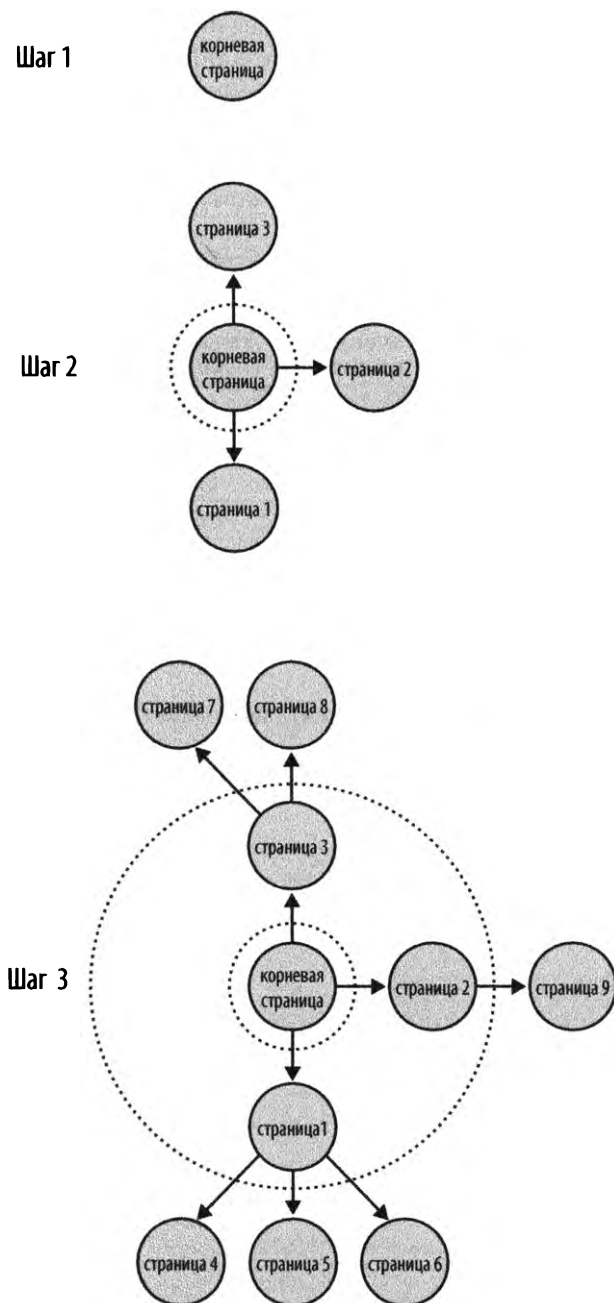


Рис. 6.1. В поиске в ширину каждый шаг увеличивает глубину на 1, и так продолжается до достижения максимальной глубины или некоторого другого критерия

Если предположить, что каждый узел в графе имеет 5 соседей и максимальная глубина равна 1, тогда вы получите 6 узлов: корневой (начальный) узел и 5 его соседей. Если каждый из этих 5 соседей также имеет 5 соседей и максимальная глубина равна 2, в конечном счете вы получите 31 узел: корневой узел, 5 соседей корневого узла и по 5 соседей каждого из соседей корневого узла. В табл. 6.1 показано, как растет величина b^d с изменением b и d .

Таблица 6.1. Влияние коэффициента ветвления графа и глубины поиска на количество узлов

Коэффициент ветвления	Узлов на глубине 1	Узлов на глубине 2	Узлов на глубине 3	Узлов на глубине 4	Узлов на глубине 5
2	3	7	15	31	63
3	4	13	40	121	364
4	5	21	85	341	1365
5	6	31	156	781	3906
6	7	43	259	1555	9331

На рис. 6.2 изображены графики изменения величин, перечисленных в табл. 6.1.

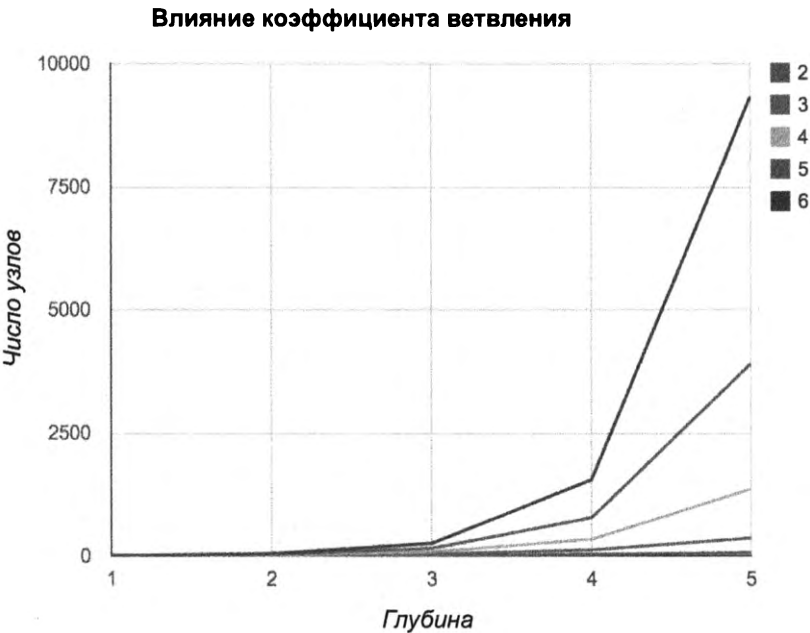


Рис. 6.2. Рост числа узлов с увеличением глубины в поиске в ширину

Предыдущие замечания в большей степени относятся к теоретическим границам алгоритма, но отметим еще одно важное обстоятельство, касающееся практической производительности алгоритма для набора данных фиксированного размера. Простейшее профилирование реализации поиска в ширину, извлекающей веб-страницы, скорее всего, покажет, что подавляющее количество времени код *тратит на операции ввода/вывода*, ожидая, пока библиотека вернет запрошенное содержимое для обработки. В подобных ситуациях, когда основное время расходуется на ввод/вывод, производительность можно увеличить, используя пул потоков выполнения¹ и загружая веб-страницы параллельно.

6.3. Определение семантики декодированием синтаксиса

Как рассказывалось в предыдущей главе, основные недостатки TF-IDF и косинусного сходства обусловлены тем, что эти модели не учитывают семантику данных и игнорируют много важного контекста. Примеры в той главе, напротив, использовали простейший синтаксический анализ, выделяя лексемы по пробельным символам, чтобы преобразовать документ в мешок слов², и простые статистические метрики сходства для определения лексем, наиболее важных для данных. Даже притом, что эти методы позволяют получать по-настоящему интересные результаты, они не дают никакого представления о значении той или иной лексемы в окружающем ее контексте внутри документа. Не будем далеко ходить за примерами: представьте предложение с омографами³, такими как *еду*, *жила* или даже *google*; один из которых может быть существительным, а другой глаголом.⁴

Обработка естественного языка (NLP) по своей природе сложна и трудна даже в относительно простых случаях, и приспособить ее для анализа широкого набора естественных языков вполне можно считать задачей века. Многие считают, что эта задача еще очень далека от решения, однако мы начинаем наблюдать рост интереса к «глубокому пониманию» Всемирной паутины благодаря по-

¹ <http://bit.ly/1a1mW5M>.

² <http://bit.ly/1a1IHDF>.

³ <http://bit.ly/1a1mWCL>.

⁴ Омонимы — частный случай омографии. Омографами являются слова, имеющие одинаковое написание. Омонимами являются слова, имеющие одинаковое написание и произношение. По каким-то причинам *омонимы* имеют большее распространение, чем *омографы*, даже при неправильном использовании.

явлению таких инициатив, как Google Knowledge Graph¹, которые продвигаются как «будущее поиска». В конце концов, полное овладение приемами NLP выглядит убедительной стратегией для преодоления теста Тьюринга², и для самого внимательного наблюдателя компьютерная программа, достигающая такого уровня «понимания», будет выглядеть как обладающая человеческим интеллектом, даже если этот мозг не биологический, а программный, реализующий математическую модель.

В отличие от структурированных или полуструктурированных источников, по сути являющихся коллекциями записей с полями, имеющими некоторое предполагаемое значение, которое можно немедленно проанализировать, в человеческих языках есть масса тонкостей и нюансов, которые следует учитывать даже при решении, казалось бы, простых задач. Например, представьте, что вам дали документ и предложили посчитать количество предложений в нем. Эту задачу легко решит любой, имеющий элементарное знание грамматики языка, но, чтобы ту же задачу решила машина, ей необходим сложный и подробный набор инструкций.

Впрочем, обнадеживает, что машины способны быстро определять окончания предложений в четко сформированных данных почти с идеальной точностью. Но даже точно определив все предложения, вы можете почти ничего не знать о способах использования слов или фраз в этих предложениях. Яркими примерами могут служить сарказм и другие формы иронии. Даже обладая полной информацией о структуре предложения, часто требуется дополнительный контекст, окружающий предложение, чтобы правильно его интерпретировать.

Обобщая в самом широком смысле, можно сказать, что основная идея NLP заключается в том, чтобы взять документ, состоящий из упорядоченной коллекции символов, следующих правилам *синтаксиса* и *грамматики*, и определить *семантику*, связанную с этими символами.

Вернемся к задаче определения предложений, которая является первым шагом в большинстве конвейеров NLP, чтобы продемонстрировать некоторые сложности, возникающие при анализе естественного языка. По недопониманию легко переоценить полезность простой эвристики, основанной на правилах, поэтому важно внимательно изучить пример, чтобы вы могли осознать некоторые основные проблемы и не тратили время на изобретение велосипеда.

¹ <http://bit.ly/2NKhEZz>.

² <http://bit.ly/1a1mZON>.

Первой вашей попыткой решить задачу мог бы стать простой поиск в тексте точек, знаков вопроса и восклицательных знаков. Это самая очевидная эвристика для начала, но она слишком грубая и обладает высокой потенциальной погрешностью. Взгляните на следующий (относительно однозначный) текст обвинения:

Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow.

Простая лексемизация текста по знакам препинания (в частности, по точкам) дает следующий результат:

```
>>> txt = "Mr. Green killed Colonel Mustard in the study with the \
... candlestick. Mr. Green is not a very nice fellow."
>>> txt.split(".")
['Mr', 'Green killed Colonel Mustard in the study with the candlestick',
'Mr', 'Green is not a very nice fellow', '']
```

Очевидно, что для выделения предложений недостаточно простого разбиения текста по точкам без учета контекста. В данном случае источником проблем является сокращение «Мг.», широко используемое в английском языке. В предыдущей главе мы узнали, что анализ n -грамм для этого фрагмента мог бы сообщить нам, что «Mr. Green» в действительности является составной лексемой, или словосочетанием. Легко представить другие крайние случаи, с трудом поддающиеся определению путем анализа словосочетаний. Забегая вперед, также отметим, что, используя тривиальную логику, нелегко определить ключевые темы предложений. Как умный человек, вы легко определите, что ключевыми темами в нашем примере могут быть «Mr. Green» (мистер Грин), «Colonel Mustard» (полковник Мастард), «the study» (кабинет) и «the candlestick» (подсвечник), но научить машину приходить к тем же выводам — сложная задача.



Прежде чем продолжить, подумайте, как бы вы написали компьютерную программу, решающую эту задачу. Это поможет вам глубже понять суть дальнейшей дискуссии.

Возможно, вы вспомните несколько очевидных возможностей, таких как определение заголовков «По Регистру Символов» с помощью регулярного выражения, использование списка распространенных сокращений и применение некоторой разновидности логики поиска границ предложений, чтобы попытаться избежать неприятностей. Да, конечно. Такой подход будет работать в некоторых ситуациях, но насколько высокой останется вероятность ошибки

для произвольного текста? Справится ли ваш алгоритм с неправильно оформленным текстом; с большим числом сокращений, как в постах на форумах или твитах; или с текстом на таких романтических языках, как испанский, французский или итальянский? В этой сфере нет простых решений, и именно поэтому анализ текста является такой важной темой в эпоху, когда объем данных на человеческом языке увеличивается буквально каждую секунду.

6.3.1. Пошаговая иллюстрация обработки естественного языка

Давайте подготовимся к пошаговому выполнению серии примеров, иллюстрирующих применение методов NLP с использованием NLTK. Далее мы рассмотрим конвейер NLP, включающий следующие шаги:

1. Определение концов предложений.
2. Лексемизация.
3. Маркировка частей речи.
4. Объединение.
5. Извлечение.



Для простой и ясной иллюстрации входных и выходных данных следующий конвейер NLP представлен в форме сеанса работы с интерпретатором Python. Однако для простоты все шаги этого конвейера реализованы в виде блокнота Jupyter Notebook для этой главы наряду с другими примерами.

Для иллюстрации продолжим использовать фрагмент текста из предыдущего раздела: «Mr. Green killed Colonel Mustard in the study with the candlestick. Mr. Green is not a very nice fellow». («Мистер Грин убил полковника Мастарда в кабинете подсвечником. Мистер Грин плохой человек».) Не забывайте, что даже если вы уже прочитали этот текст и разобрались в его грамматической структуре, для машины он остается непрозрачным строковым значением. А теперь подробнее рассмотрим шаги, которые мы должны выполнить.

Определение окончаний предложений

Этот шаг преобразует текст в коллекцию смысловых предложений. Поскольку обычно каждое предложение представляет законченную мысль, они, как правило, имеют предсказуемый синтаксис, который легко поддается дальнейшему анализу. Большинство конвейеров NLP, с которыми вы еще

встретитесь, выполняют этот первый шаг, потому что лексемизации (следующий шаг) подвергаются уже отдельные предложения. Деление текста на абзацы или разделы может повысить ценность некоторых видов анализа, но вряд ли это поможет решить задачу определения окончания предложений. Вот как можно выполнить разбор по предложениям в интерпретаторе с помощью NLTK:

```
>>> import nltk
>>> txt = "Mr. Green killed Colonel Mustard in the study with the \
... candlestick. Mr. Green is not a very nice fellow."
>>> txt = "Mr. Green killed Colonel Mustard in the study with the \
... candlestick. Mr. Green is not a very nice fellow."
>>> sentences = nltk.tokenize.sent_tokenize(txt)
>>> sentences
['Mr. Green killed Colonel Mustard in the study with the candlestick.',
 'Mr. Green is not a very nice fellow.']
```

Подробнее о происходящем за кулисами функции `sent_tokenize` мы поговорим в следующем разделе. А пока примем на веру, что она правильно обнаруживает окончания предложений в произвольном тексте, действуя более точно, не ограничиваясь анализом символов, являющихся знаками препинания.

Лексемизация

Этот шаг воздействует на отдельные предложения, разбивая их на лексемы. Следуя за примером, выполните следующие команды:

```
>>> tokens = [nltk.tokenize.word_tokenize(s) for s in sentences]
>>> tokens
[['Mr.', 'Green', 'killed', 'Colonel', 'Mustard', 'in', 'the', 'study',
 'with', 'the', 'candlestick', '.'],
 ['Mr.', 'Green', 'is', 'not', 'a', 'very', 'nice', 'fellow', '.']]
```

Обратите внимание, что в этом простом примере лексемизация дала те же результаты, что и простое разбиение по пробельным символам, только при этом были правильно опознаны символы, завершающие предложения (точки). Как будет показано в следующем разделе, этот механизм способен на большее, если дать ему такую возможность, и теперь мы уже знаем, что отличить точку, завершающую сокращение, от точки, завершающей предложение, не всегда просто. Интересно отметить, что в некоторых письменных языках, например иероглифических, лексемы в предложениях могут не отделяться пробелами, но читатель (или машина) должен различать их границы.

Маркировка частей речи

На этом шаге каждой лексеме присваивается метка, определяющая ее часть речи (Part-Of-Speech, POS). Выполните в сеансе интерпретатора еще один шаг, чтобы снабдить лексемы тегами с частями речи:

```
>>> pos_tagged_tokens = [nltk.pos_tag(t) for t in tokens]
>>> pos_tagged_tokens
[(['Mr.', 'NNP'), ('Green', 'NNP'), ('killed', 'VBD'), ('Colonel', 'NNP'),
 ('Mustard', 'NNP'), ('in', 'IN'), ('the', 'DT'), ('study', 'NN'),
 ('with', 'IN'), ('the', 'DT'), ('candlestick', 'NN'), ('.', '.')],
 [(['Mr.', 'NNP'), ('Green', 'NNP'), ('is', 'VBZ'), ('not', 'RB'),
 ('a', 'DT'), ('very', 'RB'), ('nice', 'JJ'), ('fellow', 'JJ'),
 ('.', '.')]]
```

Все эти малопонятные теги обозначают части речи. Например, 'NNP' сообщает, что лексема является существительным (noun) и является частью именной конструкции (noun phrase), 'VBD' обозначает глагол (verb) в простом прошедшем времени, а 'JJ' обозначает прилагательное (adjective). Полный перечень тегов POS¹ с их кратким описанием можно найти на сайте проекта Penn Treebank Project². После маркировки частями речи становится очевидно, насколько широкие возможности открываются для анализа. Например, используя теги POS, можно объединить существительные, являющиеся частями именных конструкций, и затем попытаться понять, какие типы сущностей они представляют (например, имена людей, названия мест или организаций). Если вам и в голову не приходило, что когда-нибудь придется на практике применять эти упражнения из программы начальной школы по определению частей речи, задумайтесь: это важно для правильной обработки естественного языка.

Объединение

Этот шаг включает анализ каждой лексемы с меткой части речи в предложении и сборку составных лексем, выражающих логические понятия — совершенно иной подход к определению словосочетаний, отличный от статистического анализа. С помощью `chunk.regexp.RegexpParser` можно определить нестандартную грамматику, но обзор этой темы выходит далеко за рамки этой главы; поэтому за подробным описанием обращайтесь к главе 9 в книге Эдварда Лопера (Edward Loper), Эвана Клейна (Ewan Klein) и Сти-

¹ <http://bit.ly/1a1n05o>.

² <http://bit.ly/2C5ecDq>.

вена Бирда (Steven Bird) *Natural Language Processing with Python* (O'Reilly)¹. В пакете NLTK имеется функция, сочетающая объединение с извлечением именованных сущностей, которое является следующим нашим шагом.

Извлечение

Этот шаг включает анализ каждого фрагмента, полученного после объединения, и маркировку этих фрагментов как именованных сущностей, таких как имена людей, названия мест, организаций и т. д. Ниже демонстрируется продолжение саги о NLP:

```
>>> ne_chunks = list(nltk.chunk.ne_chunk_sents(pos_tagged_tokens))
>>> print(ne_chunks)
[Tree('S', [Tree('PERSON', [('Mr.', 'NNP')]),
Tree('PERSON', [('Green', 'NNP')]), ('killed', 'VBD'),
Tree('ORGANIZATION', [('Colonel', 'NNP'), ('Mustard', 'NNP')]),
('in', 'IN'), ('the', 'DT'), ('study', 'NN'), ('with', 'IN'),
('the', 'DT'), ('candlestick', 'NN'), ('.', '.')]),
Tree('S', [Tree('PERSON', [('Mr.', 'NNP')]),
Tree('ORGANIZATION',
[('Green', 'NNP')]), ('is', 'VBZ'), ('not', 'RB'),
('a', 'DT'), ('very', 'RB'), ('nice', 'JJ'),
('fellow', 'JJ'), ('.', '.')])])
>>> ne_chunks[0].pprint() # каждый фрагмент можно вывести в более
# удобочитаемом виде

(S
  (PERSON Mr./NNP)
  (PERSON Green/NNP)
  killed/VBD
  (ORGANIZATION Colonel/NNP Mustard/NNP)
  in/IN
  the/DT
  study/NN
  with/IN
  the/DT
  candlestick/NN
  ./.)
```

Не стоит сейчас углубляться в детальное изучение полученного дерева. Если говорить кратко, оно объединяет некоторые лексемы и пытается классифицировать их как определенные типы сущностей. (Вы можете заметить, что составная лексема «Мг. Green» была идентифицирована как имя человека,

¹ <http://bit.ly/2szE1HW>.

но, к сожалению, лексема «Colonel Mustard» была опознана как название организации.) На рис. 6.3 показано, как выглядит вывод в Jupyter Notebook.

Несмотря на целесообразность дальнейшего изучения приемов анализа естественного языка с помощью пакета NLTK, это не является нашей целью. Примеры в этом разделе призваны лишь показать уровень сложности задачи и побудить вас прочитать книгу о NLTK¹ или заглянуть на один из многочисленных ресурсов в интернете, чтобы продолжить исследование этой темы.

Учитывая широкие возможности настройки разных аспектов NLTK, в оставшейся части главы будет предполагаться использование NLTK с настройками по умолчанию, если явно не оговаривается иное.

А теперь, закончив краткое введение в NLP, попробуем проанализировать некоторые данные из блога.

```
In [34]: # Downloading nltk packages used in this example
nltk.download('maxent_ne_chunker')
nltk.download('words')

ne_chunks = list(nltk.chunk.ne_chunk_sents(pos_tagged_tokens))
print(ne_chunks)
ne_chunks[0].pprint()

[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data] /Users/mikhail/nltk_data...
[nltk_data] Package maxent_ne_chunker is already up-to-date!
[nltk_data] Downloading package words to /Users/mikhail/nltk_data...
[nltk_data] Package words is already up-to-date!
[Tree('S', [Tree('PERSON', [(('Mr.', 'NNP')])], Tree('PERSON', [(('Green', 'NNP')])], ('killed', 'VBD'), Tree('ORGANIZATION', [(('Colonel', 'NNP'), ('Mustard', 'NNP')])], ('in', 'IN'), ('the', 'DT'), ('study', 'NN'), ('with', 'IN'), ('the', 'DT'), ('candlestick', 'NN'), (('.', '.'))], Tree('S', [Tree('PERSON', [(('Mr.', 'NNP')])], Tree('ORGANIZATION', [(('Green', 'NNP')])], ('is', 'VBS'), ('not', 'RB'), ('a', 'DT'), ('very', 'RB'), ('nice', 'JJ'), ('fellow', 'NN'), (('.', '.')))]))
(S
 (PERSON Mr./NNP)
 (PERSON Green/NNP)
 killed/VBD
 (ORGANIZATION Colonel/NNP Mustard/NNP)
 in/IN
 the/DT
 study/NN
 with/IN
 the/DT
 candlestick/NN
 ./.))
```

Рис. 6.3. Объединение лексем и их классификация с помощью NLTK

6.3.2. Выделение предложений из данных на человеческом языке

Строительство своих стеков NLP вы часто будете начинать с выделения предложений, поэтому начнем с решения этой задачи. Даже если вы решите отказать от всех остальных задач, определение конца предложений откроет

¹ <http://bit.ly/1a1mtAk>

перед вами достаточно широкие возможности, включая обобщение документов, которое рассматривается в следующем разделе. Но прежде попробуем получить очищенные данные на человеческом языке. Используем проверенный пакет `feedparser`, а также некоторые утилиты, представленные в предыдущей главе, основанные на `nlk` и `BeautifulSoup`, извлечем с их помощью несколько статей из блога O'Reilly Ideas¹ и очистим их от разметки HTML. Код в примере 6.4 извлекает несколько статей и сохраняет их в локальном файле в формате JSON.

Пример 6.4. Извлечение данных из блога путем парсинга лент новостей

```
import os
import sys
import json
import feedparser
from bs4 import BeautifulSoup
from nltk import clean_html

FEED_URL = 'http://feeds.feedburner.com/oreilly/radar/atom'

def cleanHtml(html):
    if html == "": return ""

    return BeautifulSoup(html, 'html5lib').get_text()

fp = feedparser.parse(FEED_URL)

print("Fetched {0} entries from '{1}'".format(len(fp.entries[0].title),
      fp.feed.title))

blog_posts = []
for e in fp.entries:
    blog_posts.append({'title'   : e.title,
                      'content' : cleanHtml(e.content[0].value),
                      'link'    : e.links[0].href})

out_file = os.path.join('feed.json')
f = open(out_file, 'w+')
f.write(json.dumps(blog_posts, indent=1))
f.close()

print('Wrote output file to {0}'.format(f.name))
```

Загрузив данные на человеческом языке из авторитетного источника, мы получили роскошь положиться на хорошую грамматику; это также дает надежду,

¹ <http://oreil.ly/2QwxDch>.

что один из механизмов определения границ предложений в NLTK идеально справится со своей задачей. Нет лучшего способа узнать, что произойдет, чем опробовать некоторый код, поэтому двинемся вперед и рассмотрим листинг в примере 6.5. Он использует методы `sent_tokenize` и `word_tokenize`, которые являются псевдонимами механизмов NLTK, реализующих выделение предложений и лексемизацию слов. Вслед за листингом последует краткое его пояснение.

Пример 6.5. Использование инструментов NLP из пакета NLTK для обработки данных из блога

```
import json
import nltk

BLOG_DATA = "resources/ch06-webpages/feed.json"

blog_data = json.loads(open(BLOG_DATA).read())

# Загрузить пакеты nltk, используемые в этом примере
nltk.download('stopwords')

# При необходимости настройте свой список стоп-слов. Здесь добавляются
# некоторые распространенные сокращения и знаки препинания.

stop_words = nltk.corpus.stopwords.words('english') + [
    ',',
    ';',
    ':',
    '-',
    '\s',
    '?',
    ')',
    '(',
    ':',
    '\'',
    '\re',
    '"',
    '-',
    '}',
    '{',
    u' - ',
    ']',
    '[',
    '...',
]

for post in blog_data:
    sentences = nltk.tokenize.sent_tokenize(post['content'])
    words = [w.lower() for sentence in sentences for w in
```

```

nltk.tokenize.word_tokenize(sentence)]

fdist = nltk.FreqDist(words)

# Удалить стоп-слова из fdist
for sw in stop_words:
    del fdist[sw]

# Вычислить простые статистики

num_words = sum([i[1] for i in fdist.items()])
num_unique_words = len(fdist.keys())

# Гапаксы (hapaxes) -- это слова, встречаемые единожды
num_hapaxes = len(fdist.hapaxes())

top_10_words_sans_stop_words = fdist.most_common(10)

print(post['title'])
print('\tNum Sentences:'.ljust(25), len(sentences))
print('\tNum Words:'.ljust(25), num_words)
print('\tNum Unique Words:'.ljust(25), num_unique_words)
print('\tNum Hapaxes:'.ljust(25), num_hapaxes)
print('\tTop 10 Most Frequent Words (sans stop words):\n\t\t',
      '\n\t\t'.join(['{0} ({1})'.format(w[0], w[1])
                    for w in top_10_words_sans_stop_words]))
print()

```

В NLTK имеется несколько механизмов лексемизации, но в документации к пакету, как лучшие из имеющихся, «рекомендуется» использовать псевдонимы `sent_tokenize` и `word_tokenize`. На момент написания этих строк (вы можете убедиться в этом сами с помощью `pydoc` или команды `nltk.tokenize.sent_tokenize?` в IPython или Jupyter Notebook) под этими псевдонимами скрывались `PunktSentenceTokenizer` и `TreebankWordTokenizer` соответственно. Давайте кратко рассмотрим их.

Внутренне `PunktSentenceTokenizer` в значительной степени опирается на возможность определения сокращений, входящих в состав словосочетаний, и в парсинге предложений использует некоторые регулярные выражения, учитывающие некоторые типичные шаблоны использования знаков препинания. Полное описание логики работы `PunktSentenceTokenizer` выходит далеко за рамки этой книги, поэтому мы рекомендуем найти время и прочитать статью Тибора Кисса (Tibor Kiss) и Яна Странка (Jan Strunk) под названием «Unsupervised Multilingual Sentence Boundary Detection»¹, где все эти детали описаны простым и понятным языком.

¹ <http://bit.ly/2EzWCEZ>.

Как будет показано, мы можем создать экземпляр `PunktSentenceTokenizer` с примером текста для его обучения, чтобы повысить его точность. В основе класса лежит *алгоритм обучения без учителя*; он не требует явной разметки обучающего фрагмента данных. Вместо этого алгоритм исследует некоторые *признаки*, присутствующие в самом тексте, такие как символы в верхнем регистре и совместное появление лексем, и на их основе определяет наиболее подходящие параметры, управляющие разбиением текста на предложения.

Самый простой механизм лексемизации в NLTK — класс `WhitespaceTokenizer`, который разбивает текст на лексемы по пробельным символам. Но, как вы уже знаете, это решение имеет ряд недостатков. Поэтому в настоящее время NLTK рекомендует использовать `TreebankWordTokenizer` — механизм лексемизации, который обрабатывает предложения и использует соглашения, устанавливаемые проектом Penn Treebank Project (<http://bit.ly/2C5ecDq>).¹ Единственное, что может заставить вас врасплох — в процессе лексемизации `TreebankWordTokenizer` выполняет некоторые неочевидные действия, такие как раздельная маркировка компонентов составных слов и существительных в притяжательном падеже. Например, парсинг предложения «I'm hungry» вернет отдельные компоненты «I» и «'m», сохранив различия между сказуемым и подлежащим из двух слов, объединенных в составное слово «I'm». Нетрудно понять, что такое дробление грамматической информации может оказаться весьма кстати, когда придет время провести расширенный анализ, устанавливающий отношения между подлежащими и сказуемыми в предложениях.

С помощью механизмов выделения предложений и лексемизации можно сначала разбить текст на предложения, а затем каждое предложение разбить на лексемы. Несмотря на простоту этого подхода, он имеет важный недостаток — ошибки, вызванные неправильным выделением предложений, повлияют на результаты дальнейшей лексемизации и потенциально могут отрицательно сказаться на общей точности остальной части стека NLP. Например, если механизм выделения предложений ошибочно решит, что точка после «Mr.» отмечает конец предложения в тексте «Mr. Green killed Colonel Mustard in the study with the candlestick», в дальнейшем окажется невозможно извлечь сущность «Mr. Green» из текста без применения специализированной логики. И снова многое зависит от сложности полного стека NLP и от того, как он управляет распространением ошибок.

¹ *Treebank* — очень интересный термин, он обозначает корпус с расширенной лингвистической информацией. Такое название (слово *treebank* можно перевести как «банк деревьев». — *Примеч. пер.*) выбрано специально, чтобы подчеркнуть, что этот банк (или коллекция) содержит предложения в виде деревьев, соответствующих конкретной грамматике.

По умолчанию класс `PunktSentenceTokenizer` обучен на данных в корпусе Penn Treebank и прекрасно справляется со своей задачей. Конечной целью парсинга является создание объекта `nltk.FreqDist` (его можно считать чуть более сложной версией `collections.Counter`), который принимает список лексем. Остальной код в примере 6.5 просто демонстрирует применение некоторых часто используемых методов NLTK.



Если вы испытываете проблемы с продвинутыми механизмами парсинга в NLTK, такими как `TreebankWordTokenizer` или `PunktWordTokenizer`, используйте `WhitespaceTokenizer`, пока не решите, какой из более продвинутых механизмов задействовать. В действительности использование более простого решения часто может быть даже выгодным. Например, применение продвинутого механизма для парсинга текста, содержащего много адресов URL, может оказаться не лучшим решением.

Целью этого раздела было познакомить вас с первым шагом на пути создания конвейера NLP. Попутно мы разработали несколько метрик, предприняв слабую попытку охарактеризовать некоторые данные из блога. Наш конвейер не предусматривает ни маркировки частей речи, ни объединения лексем (пока), но уже дает представление о некоторых понятиях и заставляет задуматься о некоторых сопутствующих проблемах. Мы могли бы просто выделить лексемы по пробелам, подсчитать их и получить немало полезной информации, но вскоре вы по достоинству оцените выбранный нами более сложный путь, ведущий к более глубокому пониманию данных. Для иллюстрации одного из возможных применений только что полученных знаний в следующем разделе мы рассмотрим простой алгоритм обобщения документов, который опирается только на сегментацию предложений и частотный анализ.

6.3.3. Обобщение документов

Способность подходов NLP довольно качественно разграничивать предложения в неструктурированном тексте открывает ряд весьма мощных возможностей анализа текста, таких как грубое, но довольно эффективное обобщение документов. Существует много способов и подходов, и один из самых простых, с которых все начиналось, был описан в апрельском номере *IBM Journal* за 1958 год. В основополагающей статье, озаглавленной «The Automatic Creation of Literature Abstracts»¹, Х. П. Лун (H.P. Luhn) описал прием, который факти-

¹ <http://bit.ly/1a1n4Cj>.

чески сводится к фильтрации предложений, содержащих часто встречающиеся слова, появляющиеся по соседству друг с другом.

Оригинальная статья написана простым и понятным языком и очень интересна; Лун фактически описывает, как он готовил перфокарты для выполнения тестов с разными параметрами! Просто удивительно — то, что сейчас можно реализовать десятком строк кода на Python и выполнить на недорогом компьютере, в то время требовало нескольких часов упорной работы, чтобы запрограммировать гигантский мейнфрейм. В примере 6.6 приводится упрощенная реализация алгоритма Луна для обобщения документов, а краткий его анализ — в следующем разделе. Прежде чем перейти к обсуждению, изучите код и постарайтесь понять, как он работает.



Пример 6.6 использует пакет `numpy` (коллекцию оптимизированных численных операций), который должен быть установлен вместе с `nltk`. Если по какой-то причине вы решили не использовать виртуальную машину и этот пакет отсутствует в вашей системе, просто выполните команду `pip install numpy`.

Пример 6.6. Алгоритм обобщения документа, основанный на выделении предложений и частотном анализе

```
import json
import nltk
import numpy

BLOG_DATA = "resources/ch06-webpages/feed.json"

blog_data = json.loads(open(BLOG_DATA).read())

N = 100 # Число рассматриваемых слов
CLUSTER_THRESHOLD = 5 # Учитываемое расстояние между словами
TOP_SENTENCES = 5 # Число предложений, составляющих обобщение

# Расширенный список стоп-слов
stop_words = nltk.corpus.stopwords.words('english') + [
    '.',
    ',',
    '!',
    '...',
    '\s',
    '?',
    ')',
    '(',
    ':',
    '\'',
    '\re',
```

```
'"',
'-',
'}',
{'',
u' - ',
'>',
'<',
'...'
]
```

```
# Подход, предпринятый Х.П. Луном
# в статье "The Automatic Creation of Literature Abstracts"
def _score_sentences(sentences, important_words):
    scores = []
    sentence_idx = 0

    for s in [nltk.tokenize.word_tokenize(s) for s in sentences]:
        word_idx = []

        # Для каждого слова в списке слов...
        for w in important_words:
            try:
                # Определить позиции любых важных слов
                # в предложении
                word_idx.append(s.index(w))
            except ValueError: # w not in this particular sentence
                pass

        word_idx.sort()

        # Некоторые предложения могут вообще не содержать важных слов
        if len(word_idx) == 0: continue

        # Используя позиции, определить кластеры на основе предельного расстояния
        # для любых двух соседствующих слов

        clusters = []
        cluster = [word_idx[0]]
        i = 1
        while i < len(word_idx):
            if word_idx[i] - word_idx[i - 1] < CLUSTER_THRESHOLD:
                cluster.append(word_idx[i])
            else:
                clusters.append(cluster[:])
                cluster = [word_idx[i]]
            i += 1
        clusters.append(cluster)

    # Оценить каждый кластер. Выбрать максимальную оценку кластера
```

```

# в качестве оценки для всего предложения.
max_cluster_score = 0

for c in clusters:
    significant_words_in_cluster = len(c)
    # Кластеры также содержат неважные слова, поэтому мы определяем общую
    # длину кластера по позициям слов
    total_words_in_cluster = c[-1] - c[0] + 1
    score = 1.0 * significant_words_in_cluster**2 / total_words_in_cluster

    if score > max_cluster_score:
        max_cluster_score = score

    scores.append((sentence_idx, max_cluster_score))
    sentence_idx += 1

return scores

def summarize(txt):
    sentences = [s for s in nltk.tokenize.sent_tokenize(txt)]
    normalized_sentences = [s.lower() for s in sentences]

    words = [w.lower() for sentence in normalized_sentences for w in
              nltk.tokenize.word_tokenize(sentence)]

    fdist = nltk.FreqDist(words)

    # Удалить стоп-слова из fdist
    for sw in stop_words:
        del fdist[sw]

    top_n_words = [w[0] for w in fdist.most_common(N)]

    scored_sentences = _score_sentences(normalized_sentences, top_n_words)

    # Обобщение, подход 1:
    # Отбросить неважные предложения, используя как фильтр среднюю оценку
    # плюс долю стандартного отклонения

    avg = numpy.mean([s[1] for s in scored_sentences])
    std = numpy.std([s[1] for s in scored_sentences])
    mean_scored = [(sent_idx, score) for (sent_idx, score) in scored_sentences
                    if score > avg + 0.5 * std]

    # Обобщение, подход 2:
    # Другое решение -- вернуть только N предложений с самыми высокими оценками

    top_n_scored = sorted(scored_sentences, key=lambda s: s[1])[-TOP_SENTENCES:]
    top_n_scored = sorted(top_n_scored, key=lambda s: s[0])

```

```
# Дополнить объект документа обобщениями

return dict(top_n_summary=[sentences[idx] for (idx, score) in top_n_scored],
            mean_scored_summary=[sentences[idx] for (idx, score)
                                in mean_scored])

blog_data = json.loads(open(BLOG_DATA).read())

for post in blog_data:

    post.update(summarize(post['content']))

    print(post['title'])
    print('=' * len(post['title']))
    print()
    print('Top N Summary')
    print('-----')
    print(' '.join(post['top_n_summary']))
    print()
    print('Mean Scored Summary')
    print('-----')
    print(' '.join(post['mean_scored_summary']))
    print()
```

В качестве примера используем статью Тима О’Рейли (Tim O’Reilly) «The Louvre of the Industrial Age»¹. Она содержит примерно 460 слов, и здесь приводится полный ее текст, чтобы можно было сравнить результаты обобщения двумя способами:

This morning I had the chance to get a tour of The Henry Ford Museum in Dearborn, MI, along with Dale Dougherty, creator of Make: and Makerfaire, and Marc Greuther, the chief curator of the museum. I had expected a museum dedicated to the auto industry, but it’s so much more than that. As I wrote in my first stunned tweet, “it’s the Louvre of the Industrial Age.”

When we first entered, Marc took us to what he said may be his favorite artifact in the museum, a block of concrete that contains Luther Burbank’s shovel, and Thomas Edison’s signature and footprints. Luther Burbank was, of course, the great agricultural inventor who created such treasures as the nectarine and the Santa Rosa plum. Ford was a farm boy who became an industrialist; Thomas Edison was his friend and mentor. The museum, opened in 1929, was Ford’s personal homage to the transformation of the world that he was so much a part of. This museum chronicles that transformation.

¹ <http://oreil.ly/1a1n4SO>.

The machines are astonishing — steam engines and coal-fired electric generators as big as houses, the first lathes capable of making other precision lathes (the makerbot of the 19th century), a ribbon glass machine that is one of five that in the 1970s made virtually all of the incandescent lightbulbs in the world, combine harvesters, railroad locomotives, cars, airplanes, even motels, gas stations, an early McDonalds' restaurant and other epiphenomena of the automobile era.

Under Marc's eye, we also saw the transformation of the machines from purely functional objects to things of beauty. We saw the advances in engineering — the materials, the workmanship, the design, over a hundred years of innovation. Visiting The Henry Ford, as they call it, is a truly humbling experience. I would never in a hundred years have thought of making a visit to Detroit just to visit this museum, but knowing what I know now, I will tell you confidently that it is as worth your while as a visit to Paris just to see the Louvre, to Rome for the Vatican Museum, to Florence for the Uffizi Gallery, to St. Petersburg for the Hermitage, or to Berlin for the Pergamon Museum. This is truly one of the world's great museums, and the world that it chronicles is our own.

I am truly humbled that the Museum has partnered with us to hold Makerfaire Detroit on their grounds. If you are anywhere in reach of Detroit this weekend, I heartily recommend that you plan to spend both days there. You can easily spend a day at Makerfaire, and you could easily spend a day at The Henry Ford.

P. S. Here are some of my photos from my visit. (More to come soon. Can't upload many as I'm currently on a plane.)

Перевод текста статьи на русский язык:

Сегодня утром мне представилась возможность совершить экскурсию в музей Генри Форда в Дирборне, штат Мичиган, вместе с Дейлом Догерти (Dale Dougherty), основателем Make: и Makerfaire, и Марком Гройтером (Marc Greuther), главным хранителем музея. Я ожидал увидеть музей истории автомобильной промышленности, но его экспозиция оказалась намного шире. Как я уже писал в своем твите: «Это Лувр индустриальной эпохи».

Войдя в музей, Марк повел нас к самому популярному, по его словам, экспонату музея — бетонному блоку с лопатой Лютера Бербанка (Luther Burbank) и росписью и отпечатком ноги Томаса Эдисона (Thomas Edison). Лютер Бербанк был известным селекционером. Он вывел такие шедевры,

как нектарин и слива Санта Роза (Santa Rosa). Форд рос и воспитывался на ферме, прежде чем стал промышленником; Томас Эдисон был его другом и наставником. Музей, открытый в 1929 году, стал данью уважения Форда к преобразованию мира, частью которого он являлся. Музей стал летописью этих преобразований.

Машины, выставленные в музее, поражают воображение — огромные, как дома, паровые двигатели и электрогенераторы на угле, первые токарные станки, благодаря которым стало возможным создать другие, высокоточные токарные станки (произведенные в конце XIX века), стеклоформирующая машина конвейерного типа — одна из пяти, которые в 1970-х производили чуть ли не все лампы накаливания в мире, комбайны, железнодорожные локомотивы, автомобили, самолеты и даже мотели, заправочные станции, один из первых ресторанов «Макдоналдс» и другие явления автомобильной эпохи.

Следуя за Марком, мы увидели, как машины превращались из чисто утилитарных вещей в предметы красоты. Мы увидели, как развивалось машиностроение в течение сотни лет — изменялись материалы, качество изготовления, дизайн. Посещение «Генри Форда», как они называют свой музей, произвело на меня по-настоящему неизгладимое впечатление. Сам я никогда не додумался бы приехать в Детройт только ради посещения этого музея, но теперь я с уверенностью могу сказать, что он стоит того, так же как поездка в Париж только ради посещения Лувра, в Рим — ради музея Ватикана, во Флоренцию — ради галереи Уффици, в Санкт-Петербург — ради Эрмитажа или в Берлин — ради Пергамского музея. Это действительно один из величайших музеев мира, и рассказывает он о нашем собственном мире.

Считаю для себя большой честью, что музей пошел нам навстречу и разрешил провести фестиваль Makerfaire в Детройте на их территории. Если в эти выходные вы будете неподалеку от Детройта, я от всей души рекомендую вам провести оба дня именно там. Вы могли бы один день провести на фестивале Makerfaire, а другой посвятить посещению музея Генри Форда.

P. S. Вот несколько фотографий с экскурсии. (Скоро будут еще. Не могу выложить сразу много, потому что сейчас лечу в самолете.)

После фильтрации предложений по среднему и стандартному отклонению в получившемся обобщении осталось около 170 слов:

This morning I had the chance to get a tour of The Henry Ford Museum in Dearborn, MI, along with Dale Dougherty, creator of Make: and Makerfaire, and Marc Greuther, the chief curator of the museum. I had expected a museum dedicated to the auto industry, but it's so much more than that. As I wrote in my first stunned tweet, "it's the Louvre of the Industrial Age. This museum chronicles that transformation. The machines are astonishing — steam engines and coal fired electric generators as big as houses, the first lathes capable of making other precision lathes (the makerbot of the 19th century), a ribbon glass machine that is one of five that in the 1970s made virtually all of the incandescent lightbulbs in the world, combine harvesters, railroad locomotives, cars, airplanes, even motels, gas stations, an early McDonalds' restaurant and other epiphenomena of the automobile era. You can easily spend a day at Makerfaire, and you could easily spend a day at The Henry Ford.

Перевод результата на русский язык:

Сегодня утром мне представилась возможность совершить экскурсию в музей Генри Форда в Дирборне, штат Мичиган, вместе с Дейлом Догерти (Dale Dougherty), основателем Make: и Makerfaire, и Марком Гройтером (Marc Greuther), главным хранителем музея. Я ожидал увидеть музей истории автомобильной промышленности, но его экспозиция оказалась намного шире. Как я уже писал в своем твите: «Это Лувр индустриальной эпохи». Музей стал летописью этих преобразований. Машины, выставленные в музее, поражают воображение — огромные, как дома, паровые двигатели и электрогенераторы на угле, первые токарные станки, благодаря которым стало возможным создать другие, высокоточные токарные станки (произведенные в конце XIX века), стеклоформирующая машина конвейерного типа — одна из пяти, которые в 1970-х производили чуть ли не все лампы накаливания в мире, комбайны, железнодорожные локомотивы, автомобили, самолеты и даже мотели, заправочные станции, один из первых ресторанов «Макдоналдс» и другие явления автомобильной эпохи. Вы могли бы один день провести на фестивале Makerfaire, а другой посвятить посещению музея Генри Форда.

Альтернативный подход к обобщению, который оставляет только N предложений с самой высокой оценкой (в данном случае $N = 5$), дал более компактный результат, включающий примерно 90 слов, и вместе с тем более информативный:

This morning I had the chance to get a tour of The Henry Ford Museum in Dearborn, MI, along with Dale Dougherty, creator of Make: and Makerfaire,

and Marc Greuther, the chief curator of the museum. I had expected a museum dedicated to the auto industry, but it's so much more than that. As I wrote in my first stunned tweet, "it's the Louvre of the Industrial Age. This museum chronicles that transformation. You can easily spend a day at Makerfaire, and you could easily spend a day at The Henry Ford.

Перевод результата на русский язык:

Сегодня утром мне представилась возможность совершить экскурсию в музей Генри Форда в Дирборне, штат Мичиган, вместе с Дейлом Догерти (Dale Dougherty), основателем Make: и Makerfaire, и Марком Гройтером (Marc Greuther), главным хранителем музея. Я ожидал увидеть музей истории автомобильной промышленности, но его экспозиция оказалась намного шире. Как я уже писал в своем твите: «Это Лувр индустриальной эпохи». Музей стал летописью индустриальных преобразований. Вы могли бы один день провести на фестивале Makerfaire, а другой посвятить посещению музея Генри Форда.

Как в любой другой ситуации, связанной с анализом, сравнивая полный текст с его обобщенной версией, можно сделать немало интересных выводов.

Мы легко можем организовать вывод в виде простой разметки, которую можно открыть в любом веб-браузере, просто добавив в конец сценария, который выводит строки, инструкции, выполняющие подстановку строк. Пример 6.7 иллюстрирует один из подходов к отображению обобщенной версии документа, который выводит полный текст статьи и выделяет жирным шрифтом предложения, составляющие обобщенную версию, чтобы проще было увидеть, какие строки вошли в обобщение, а какие — нет. Сценарий сохраняет на диск набор HTML-файлов, чтобы потом их можно было открыть в сеансе Jupyter Notebook или в браузере.

Пример 6.7. Визуализация результатов обобщения документа в виде разметки HTML

```
import os
from IPython.display import IFrame
from IPython.core.display import display

HTML_TEMPLATE = """<html>
<head>
    <title>{0}</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
</head>
<body>{1}</body>
```

```
</html>"""
```

```
for post in blog_data:
```

```
    # Использовать прежде определенную функцию обобщения
```

```
    post.update(summarize(post['content']))
```

```
    # Также можно сохранить полную версию статьи с ключевыми предложениями,
```

```
    # отмеченными для анализа с помощью простой подстановки строк
```

```
    for summary_type in ['top_n_summary', 'mean_scored_summary']:
```

```
        post[summary_type + '_marked_up'] = '<p>{0}</p>'.format(post['content'])
```

```
        for s in post[summary_type]:
```

```
            post[summary_type + '_marked_up'] =
```

```
            post[summary_type + '_marked_up'].replace(s,
```

```
            '<strong>{0}</strong>'.format(s))
```

```
    filename = post['title'].replace("?", "") + '.summary.' + summary_type +  
    '.html'
```

```
    f = open(os.path.join(filename), 'wb')
```

```
    html = HTML_TEMPLATE.format(post['title'] + ' Summary',
```

```
    post[summary_type + '_marked_up'])
```

```
    f.write(html.encode('utf-8'))
```

```
    f.close()
```

```
    print("Data written to", f.name)
```

```
# Отображение любого из этих файлов в плавающем фрейме. Здесь отображается
```

```
# последний обработанный файл, соответствующий последнему значению f.name
```

```
print()
```

```
print("Displaying {0}: ".format(f.name))
```

```
display(IFrame('files/{0}'.format(f.name), '100%', '600px'))
```

Результатом является полный текст документа, в котором предложения, попавшие в обобщенный вариант, выделены жирным шрифтом, как показано на рис. 6.4. При анализе альтернативных вариантов обобщения простое сравнение результатов в разных вкладках браузера поможет вам понять сходства и различия между ними. Главное отличие, которое можно наблюдать в данном случае, — довольно длинное (и содержательное) предложение в середине документа, начинающееся словами «The machines are astonishing» («Машины, выставленные в музее, поражают воображение»).

В следующем разделе кратко обсуждается подход к обобщению документов, предложенный Луном.

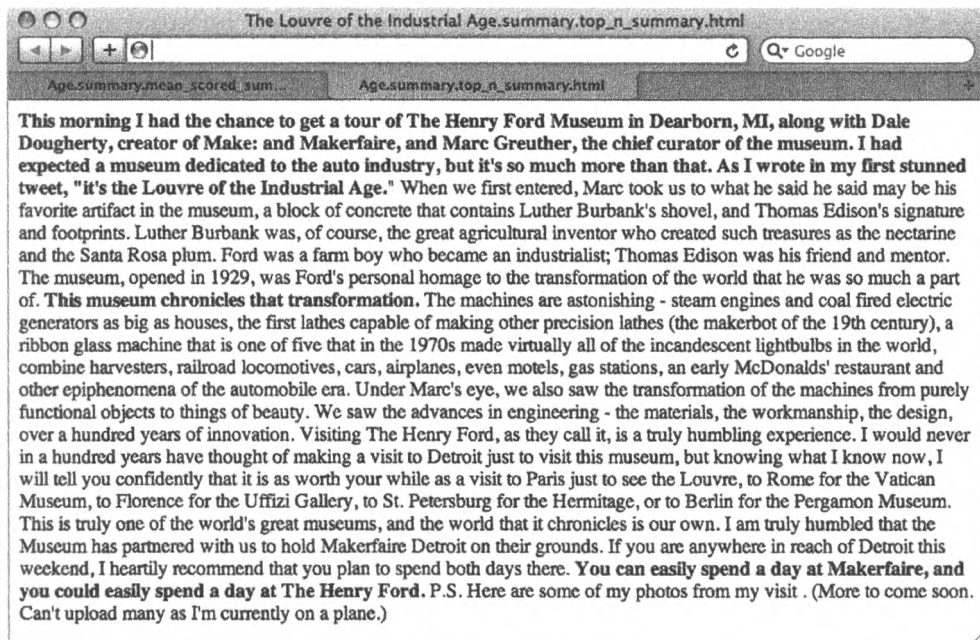


Рис. 6.4. Текст статьи из блога O'Reilly Radar с наиболее важными предложениями, согласно алгоритму обобщения выделенными жирным шрифтом

Анализ алгоритма обобщения Луна



В этом разделе представлен анализ алгоритма обобщения Луна. Цель этой дискуссии — дать вам более широкое представление о методах обработки данных на человеческом языке, но, безусловно, сведения, что приводятся здесь, не являются обязательными для успешного анализа социальных сетей. Если вы почувствуете, что запутались в деталях, просто пропустите этот раздел и вернитесь к нему позже.

Алгоритм Луна основывается на предположении, что важные предложения в документе содержат слова, встречающиеся наиболее часто. Однако есть несколько тонкостей, которые следует отметить. Во-первых, не все часто встречающиеся слова важны; стоп-слова, к примеру, являются словами-паразитами и едва ли представляют интерес для анализа. Обратите внимание, что в данном примере отфильтровываются наиболее типичные стоп-слова, но вообще можно создать и использовать свой список стоп-слов для любого конкретного блога или области, что может повысить эффективность этого или любого другого алгоритма, предполагающего удаление стоп-слов. Например, в блоге, посвященном

бейсболу, слово *бейсбол* может встречаться настолько часто, что есть смысл включить его в список стоп-слов, даже притом, что оно не является типичным стоп-словом. (Также было бы интересно внедрить метрику TF-IDF в функцию оценки для конкретного источника данных, чтобы определить наиболее общие слова для данной области.)

Следующим шагом после успешного исключения стоп-слов является выбор разумного значения N и извлечение N наиболее важных слов, которые послужат основой для дальнейшего анализа. Как предполагает алгоритм, эти N наиболее важных слов являются достаточно описательными, чтобы охарактеризовать суть документа, и из двух любых предложений в документе наиболее описательным является то, что содержит больше этих слов. После определения «важных слов» остается только применить эвристику к каждому предложению и выделить подмножество предложений для использования в качестве реферата документа. Оценка предложений осуществляется функцией `score_sentences`. Именно она выполняет львиную долю работы в примере.

Для оценки предложений функция `score_sentences` объединяет лексемы в кластеры, применяя простую метрику порогового расстояния, и оценивает каждый кластер по следующей формуле:

$$\frac{(\text{важных слов в кластере})^2}{\text{всего слов в кластере}}.$$

За оценку всего предложения принимается наибольшая из оценок кластеров, выявленных в этом предложении. Рассмотрим основные шаги, выполняемые функцией `score_sentences`, на примере предложения, чтобы лучше понять, как работает этот прием:

Вход: пример предложения

```
['Mr.', 'Green', 'killed', 'Colonel', 'Mustard', 'in', 'the',  
'study', 'with', 'the', 'candlestick', '.']
```

Вход: список важных слов

```
['Mr.', 'Green', 'Colonel', 'Mustard', 'candlestick']
```

Вход/допущение: порог объединения в кластеры (расстояние)

Промежуточные вычисления: выявленные кластеры

```
[ ['Mr.', 'Green', 'killed', 'Colonel', 'Mustard'], ['candle stick'] ]
```

Промежуточные вычисления: оценки кластеров

```
[ 3.2, 1 ] # Определяются, как: [ (4*4)/5, (1*1)/1 ]
```

Выход: оценка предложения

```
3.2 # max([3.2, 1])
```

Функция `score_sentences` просто выполняет рутинные операции, выявляя кластеры в предложении. *Кластер* определяется как последовательность, содержащая два или более важных слова, при этом каждое важное слово находится от ближайшего соседа не дальше порогового расстояния. В своей статье Лун предлагает использовать пороговое расстояние 4 или 5, но мы использовали 3, чтобы упростить пример; благодаря этому расстояние между 'Green' и 'Colonel' было оценено как достаточно большое и в первый выявленный кластер было помещено пять первых слов. Если бы слово *study* (кабинет) также попало в список важных слов, тогда в кластер было бы объединено все предложение (кроме завершающей точки).

После оценки всех предложений остается только определить, какие из них следует включить в обобщенную версию. В примере реализации использовано два подхода. Первый основан на фильтрации предложений по статистическому порогу, для чего вычисляются среднее и стандартное отклонение полученных оценок, а второй — просто возвращает N предложений с наибольшими оценками. Вы можете выбрать другой подход, в зависимости от характера данных, но в любом случае следует предусмотреть возможность настройки параметров для получения разумных результатов. Одна из положительных сторон подхода, возвращающего N предложений с наибольшими оценками, состоит в том, что он позволяет управлять размером обобщенной версии. Подход с вычислением среднего и стандартного отклонения потенциально может вернуть больше предложений, чем хотелось бы, если предложения в документе получают близкие оценки.

Алгоритм Луна прост в реализации и основан на предположении, что часто встречающиеся слова в документе являются наиболее описательными для него. Но имейте в виду, что подобно многим подходам, основанным на идеях классического информационного поиска, которые мы исследовали в предыдущей

главе, алгоритм Луна сам по себе не пытается проанализировать данные на более глубоком, семантическом уровне, даже притом, что использует не только идею «мешка слов». Он напрямую определяет обобщения как функцию от часто встречающихся слов и использует не очень сложный метод оценки предложений, но (как и в случае с TF-IDF) тем удивительнее, насколько хорошо он справляется с произвольно выбранными данными из блога.

Взвешивая необходимость реализации более сложного подхода, подумайте — стоит ли тратить время и силы, чтобы еще больше улучшить вполне приемлемые результаты обобщения, которые возвращает алгоритм Луна. Иногда простой эвристики вполне достаточно для достижения поставленных целей. Иногда все же действительно требуется нечто более современное. Самое сложное в таких случаях — оценка отношения затрат и преимуществ, которые будут сопровождать замену простой эвристики современным решением. Многие из нас склонны слишком оптимистично оценивать усилия, которые придется приложить.

6.4. Анализ сущностей: смена парадигмы

На протяжении этой главы не раз намекалось, что аналитические подходы с более глубоким проникновением в данные могут оказаться более эффективными, чем подходы, которые просто интерпретируют каждую лексему как абстрактный символ. Но что в действительности подразумевается под «глубоким проникновением»?

Одна из возможных интерпретаций: определение сущностей в документах и их использование в качестве основы для анализа, в противоположность анализу документов, основанному на поиске ключевых слов, или интерпретация поискового запроса как сущности определенного типа и соответствующая адаптация результатов. Возможно, вы даже не рассматривали проблему с такой стороны, но именно в этом направлении развиваются вновь появляющиеся технологии, такие как Wolfram|Alpha¹. Например, поиск по запросу «tim o'reilly» в Wolfram|Alpha вернет результаты, которые предполагают понимание, что искомая сущность является конкретной персоной; вы получите список документов, не просто содержащих искомые слова (рис. 6.5). Независимо от приема, используемого внутри для достижения цели, пользователь получит более ценный опыт, потому что результаты будут точнее соответствовать его ожиданиям.

¹ <http://bit.ly/2xtPzM7>.

 **WolframAlpha** computational intelligence.

Tim O'Reilly

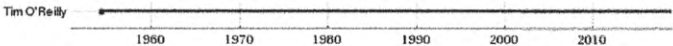
 [Browse Examples](#) [Surprise Me](#)

Input interpretation:
Tim O'Reilly (publisher, etc.) [Open code](#)

Basic information:

full name	Tim O'Reilly
date of birth	Sunday, June 6, 1954 (age: 64 years)
place of birth	Cork, Cork, Ireland

Image:


Timeline:


Familial relationships:

Parents:
Sean O'Reilly | **Anne Hillam**

Siblings:
Sean O'Reilly | **James O'Reilly** | **Anne Mary Vieira** | **Frank O'Reilly** | **Maggie Ciskanic** | **Kate Doyle**

Рис. 6.5. Пример результатов поиска в Wolfram|Alpha по запросу «tim o'reilly»

В текущем обсуждении не получится рассмотреть все разнообразие возможностей анализа сущностей, но мы вполне сможем познакомиться с приемами извлечения сущностей из документов, которые затем можно использовать для решения разных задач. Основываясь на примере конвейера NLP, представлен-

ном выше в этой главе, можно просто извлечь из документов все существительные и именные конструкции и интерпретировать их как сущности — важной основой в данном случае является предположение, что существительные и именные конструкции (или некоторые их подмножества) квалифицируются как сущности, представляющие интерес. На самом деле это вполне верное предположение и хорошая отправная точка для анализа сущностей, как показано в следующем примере. Обратите внимание, что в соответствии с соглашениями Penn Treebank, всякий тег, начинающийся с символов 'NN', определяет существительное или именную конструкцию того или иного вида. Полный список тегов Penn Treebank¹ доступен в интернете.

Пример 6.8 анализирует теги частей речи, присвоенные лексемам, и определяет существительные и именные конструкции как сущности. На языке анализа данных поиск сущностей в тексте называется *извлечением сущностей* или *распознаванием именованных сущностей*, в зависимости от нюансов решаемой задачи.

Пример 6.8. Извлечение сущностей из текста с помощью NLTK

```
import nltk
import json

BLOG_DATA = "resources/ch06-webpages/feed.json"

blog_data = json.loads(open(BLOG_DATA).read())

for post in blog_data:

    sentences = nltk.tokenize.sent_tokenize(post['content'])
    tokens = [nltk.tokenize.word_tokenize(s) for s in sentences]
    pos_tagged_tokens = [nltk.pos_tag(t) for t in tokens]

    # Преобразовать в линейный список, потому что здесь структура
    # предложений не используется и предложения гарантированно разделены
    # специальными кортежами частей речи, такими как ('.', '.')

    pos_tagged_tokens = [token for sent in pos_tagged_tokens for token in sent]

    all_entity_chunks = []
    previous_pos = None
    current_entity_chunk = []
    for (token, pos) in pos_tagged_tokens:

        if pos == previous_pos and pos.startswith('NN'):
            current_entity_chunk.append(token)
```

¹ <http://bit.ly/2obCDGA>.

```

elif pos.startswith('NN'):

    if current_entity_chunk != []:

        # Обратите внимание, что при добавлении в конец
        # current_entity_chunk может повторяться, поэтому частотный
        # анализ снова приобретает важность

        all_entity_chunks.append(' '.join(current_entity_chunk), pos))
        current_entity_chunk = [token]

    previous_pos = pos

# Сохранить объединенные конструкции как указатель для документа
# и учесть частоту...

post['entities'] = {}
for c in all_entity_chunks:
    post['entities'][c] = post['entities'].get(c, 0) + 1

# Например, можно вывести только сущности, каждое слово в которых
# начинается с символа в верхнем регистре

print(post['title'])
print('-' * len(post['title']))
proper_nouns = []

for (entity, pos) in post['entities']:
    if entity.istitle():
        print('\t{0} ({1})'.format(entity, post['entities'][(entity, pos)]))
print()

```



Как рассказывалось выше, в разделе «Пошаговая иллюстрация обработки естественного языка», в пакете NLTK имеется функция `nltk.batch_ne_chunk`, которая пытается извлечь именованные сущности из лексем с метками частей речи. Вы можете использовать эту функцию непосредственно или другие готовые модели, реализованные в NLTK.

Далее показаны довольно интересные результаты, полученные в этом примере, которые можно использовать для решения разных задач. Например, они могли бы служить превосходными подсказками для маркировки статей в таких платформах блогинга, как Word-Press:

The Louvre of the Industrial Age

 Paris (1)
 Henry Ford Museum (1)

Vatican Museum (1)
Museum (1)
Thomas Edison (2)
Hermitage (1)
Uffizi Gallery (1)
Ford (2)
Santa Rosa (1)
Dearborn (1)
Makerfaire (1)
Berlin (1)
Marc (2)
Makerfaire (1)
Rome (1)
Henry Ford (1)
Ca (1)
Louvre (1)
Detroit (2)
St. Petersburg (1)
Florence (1)
Marc Greuther (1)
Makerfaire Detroit (1)
Luther Burbank (2)
Make (1)
Dale Dougherty (1)
Louvre (1)

Статистические характеристики часто имеют разные назначения и преследуют разные цели. Обобщенные тексты предназначены для чтения, тогда как вышеприведенные списки извлеченных сущностей, позволяют быстро отыскать закономерности. Для корпусов с большими размерами, чем используемый нами в этом примере, очевидным применением списка сущностей может служить отображение облака тегов.



Попробуйте воспроизвести эти результаты, взяв текст статьи со страницы <http://oreil.ly/1a1n4SO>.

Можно ли получить такой же список терминов, слепо анализируя лексические характеристики (такие, как регистр символов) предложений? Возможно, но имейте в виду, что представленный прием позволяет также извлечь существительные и именные конструкции, не выделяемые регистром символов. Регистр действительно является важным признаком, который часто приходит на помощь, но в тексте есть другие интригующие сущности,

которые записаны только символами нижнего регистра (например, «chief curator» (главный хранитель), «locomotives» (локомотивы) и «lightbulbs» (лампы накаливания)).

Даже притом, что список сущностей не передает общего смысла текста так же эффективно, как обобщение, полученное выше, выявление сущностей может оказаться весьма ценной возможностью для анализа, поскольку эти сущности *наделены смыслом на семантическом уровне и не являются часто встречающимися словами*. На самом деле многие слова, полученные в результате извлечения сущностей, имеют довольно низкую частоту. Тем не менее они важны, потому что имеют значение для текста, а именно, они представляют людей, места, вещи или понятия, которые, как правило, являются существенной информацией.

6.4.1. Определение общего смысла данных на человеческом языке

На данный момент не трудно догадаться, что следующим важным шагом могли бы стать извлечение глаголов и определение троек (*подлежащее, сказуемое, дополнение*), чтобы узнать, какие сущности взаимодействуют друг с другом, и определить природу этих взаимодействий. Такие тройки позволяют представлять документы в виде графов объектов, которые мы могли бы просматривать намного быстрее, чем сами документы. Но самое замечательное в этом подходе — возможность взять несколько таких графов из набора документов и объединить их, чтобы определить общий смысл всего корпуса. Этот подход продолжает активно исследоваться и может найти практическое применение почти в любой ситуации, где проявляется проблема избытка информации. Но, как будет показано ниже, это мучительная проблема для общего случая и ее преодоление — задача не для слабонервных.

Если предположить, что механизм маркировки частей речи возвращает результаты в таком виде: [('Mr.', 'NNP'), ('Green', 'NNP'), ('killed', 'VBD'), ('Colonel', 'NNP'), ('Mustard', 'NNP'), ...], мы легко могли бы определить кортежи (*подлежащее, сказуемое, дополнение*) в форме: ('Mr. Green', 'killed', 'Colonel Mustard'). Однако в реальной жизни вам едва ли встретятся данные, которые имеют такую простую структуру тегов частей речи, только если вы не планируете анализировать книги для детей (кстати, неплохая отправная точка для первичной проверки своих идей). Например, рассмотрим результат маркировки пакетом NLTK первого предложения из статьи в блоге, рассма-

тривавшейся выше в этой главе, как произвольный фрагмент реалистичных данных, который понадобилось преобразовать в граф объектов:

This morning I had the chance to get a tour of The Henry Ford Museum in Dearborn, MI, along with Dale Dougherty, creator of Make: and Makerfaire, and Marc Greuther, the chief curator of the museum.

Самая простая тройка, которую можно выделить из этого предложения, — ('I', 'get', 'tour'), но, получив ее, невозможно догадаться, что Дейл Догерти (Dale Dougherty) или Марк Гройтер (Marc Greuther) тоже участвовали в экскурсии. Взглянув на данные с тегами частей речи, становится очевидно, что с их помощью очень непросто прийти к любой из этих интерпретаций, потому что предложение имеет очень сложную структуру:

```
[('This', 'DT'), ('morning', 'NN'), ('I', 'PRP'), ('had', 'VBD'),
('the', 'DT'), ('chance', 'NN'), ('to', 'TO'), ('get', 'VB'),
('a', 'DT'), ('tour', 'NN'), ('of', 'IN'), ('The', 'DT'),
('Henry', 'NNP'), ('Ford', 'NNP'), ('Museum', 'NNP'), ('in', 'IN'),
('Dearborn', 'NNP'), ('u', ',', ','), ('MI', 'NNP'), ('u', ',', ','),
('along', 'IN'), ('u', 'with', 'IN'), ('Dale', 'NNP'), ('Dougherty', 'NNP'),
('u', ',', ','), ('creator', 'NN'), ('of', 'IN'), ('Make', 'NNP'), ('u', ':', ':'),
('and', 'CC'), ('Makerfaire', 'NNP'), ('u', ',', ','), ('and', 'CC'),
('Marc', 'NNP'), ('Greuther', 'NNP'), ('u', ',', ','), ('the', 'DT'),
('chief', 'NN'), ('curator', 'NN'), ('of', 'IN'), ('the', 'DT'),
('museum', 'NN'), ('u', '.', '.')]

```

Сомнительно, что даже самый высококачественный инструментарий NLP с открытым исходным кодом смог бы в этом случае выделить значимые тройки, учитывая сложную природу предиката «had a chance to get a tour» (представилась возможность совершить экскурсию), и определить других участников экскурсии, перечисленных в конце предложения.



Желающие продолжать придерживаться стратегии конструирования троек должны уметь использовать точную информацию о тегах частей речи. Для решения продвинутых задач анализа данных на человеческом языке порой требуются значительные усилия, но результаты часто вполне оправдывают эти усилия и способны оказывать прорывное воздействие.

Даже выделяя одни лишь сущности из текста и используя их в качестве основы для анализа, как было показано выше, можно получать очень занятные результаты. Вы легко сможете выделить тройки из каждого предложения в тексте, где сказуемое в каждой тройке обозначает факт взаимодействия подлежащего с дополнением. Пример 6.9, являющийся усовершенствованной версией при-

мера 6.8, выбирает сущности из предложений, что очень может пригодиться для определения взаимодействий между сущностями с использованием предложения в качестве контекстного окна.

Пример 6.9. Выявление взаимодействий между сущностями

```
import nltk
import json

BLOG_DATA = "resources/ch06-webpages/feed.json"

def extract_interactions(txt):
    sentences = nltk.tokenize.sent_tokenize(txt)
    tokens = [nltk.tokenize.word_tokenize(s) for s in sentences]
    pos_tagged_tokens = [nltk.pos_tag(t) for t in tokens]

    entity_interactions = []
    for sentence in pos_tagged_tokens:

        all_entity_chunks = []
        previous_pos = None
        current_entity_chunk = []

        for (token, pos) in sentence:

            if pos == previous_pos and pos.startswith('NN'):
                current_entity_chunk.append(token)
            elif pos.startswith('NN'):
                if current_entity_chunk != []:
                    all_entity_chunks.append((' '.join(current_entity_chunk),
                                                         pos))
                    current_entity_chunk = [token]

            previous_pos = pos

        if len(all_entity_chunks) > 1:
            entity_interactions.append(all_entity_chunks)
        else:
            entity_interactions.append([])

    assert len(entity_interactions) == len(sentences)

    return dict(entity_interactions=entity_interactions,
                sentences=sentences)

blog_data = json.loads(open(BLOG_DATA).read())

# Вывести выбранные взаимодействия для каждого предложения
```



```

for post in blog_data:

    post.update(extract_interactions(post['content']))

    print(post['title'])
    print('-' * len(post['title']))
    for interactions in post['entity_interactions']:
        print('; '.join([i[0] for i in interactions]))
    print()

```

Полученные результаты, что приводятся ниже, подчеркивают нечто важное о природе анализа неструктурированных данных: все очень сложно!

The Louvre of the Industrial Age

morning; chance; tour; Henry Ford Museum; Dearborn; MI; Dale Dougherty; creator;
 Make; Makerfaire; Marc Greuther; chief curator

tweet; Louvre

"; Marc; artifact; museum; block; contains; Luther Burbank; shovel;
 Thomas Edison

Luther Burbank; course; inventor; treasures; nectarine; Santa Rosa

Ford; farm boy; industrialist; Thomas Edison; friend

museum; Ford; homage; transformation; world

machines; steam; engines; coal; generators; houses; lathes; precision; lathes;
 makerbot; century; ribbon glass machine; incandescent; lightbulbs; world;
 combine; harvesters; railroad; locomotives; cars; airplanes; gas; stations;
 McDonalds; restaurant; epiphenomena

Marc; eye; transformation; machines; objects; things

advances; engineering; materials; workmanship; design; years

years; visit; Detroit; museum; visit; Paris; Louvre; Rome; Vatican Museum;
 Florence; Uffizi Gallery; St. Petersburg; Hermitage; Berlin

world; museums

Museum; Makerfaire Detroit

reach; Detroit; weekend

day; Makerfaire; day

Присутствие некоторого количества шума в результатах почти неизбежно, но достижение понятных и полезных результатов (даже содержащих приемлемый уровень шума) — достойная цель. Для получения идеальных результатов, почти не содержащих шума, может потребоваться приложить огромные усилия. Однако в большинстве ситуаций это совершенно невозможно из-за врожденной сложности естественных языков и ограничений в большинстве современных инструментов, включая NLTK. Если есть возможность сделать предположение о принадлежности данных к той или иной предметной области или получить экспертные знания о природе шума, можно разработать эвристики для его эффективного устранения без риска потерять неприемлемое количество информации — но это довольно сложно.

Тем не менее взаимодействия передают некоторую ценную информацию о «сути». Например, насколько близка ваша интерпретация набора слов «morning; chance; tour; Henry Ford Museum; Dearborn; MI; Dale Dougherty; creator; Make; Makerfaire; Marc Greuther; chief curator» к оригинальному предложению?¹

Как и в предыдущем примере с обобщением, было бы полезно визуально выделить взаимодействия между сущностями. Простого изменения инструкций вывода результатов в примере 6.9, как показано в примере 6.10, достаточно, чтобы получить страницу, изображенную на рис. 6.6.

Пример 6.10. Визуализация взаимодействий между сущностями с помощью разметки HTML

```
import os
import json
import nltk
from IPython.display import IFrame
from IPython.core.display import display

BLOG_DATA = "resources/ch06-webpages/feed.json"

HTML_TEMPLATE = """<html>
    <head>
        <title>{0}</title>
```

¹ «Утром; возможность; экскурсию; музей Генри Форда; Дирборне; штат Мичиган; Дейлом Догерти; основателем; Make; Makerfaire; Марком Гройтером, главным хранителем» и исходное предложение на русском языке: «Сегодня утром мне представилась возможность совершить экскурсию в музей Генри Форда в Дирборне, штат Мичиган, вместе с Дейлом Догерти (Dale Dougherty), основателем Make: и Makerfaire, и Марком Гройтером (Marc Greuther), главным хранителем музея».

```

        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"/>
    </head>
    <body>{1}</body>
</html>""

blog_data = json.loads(open(BLOG_DATA).read())

for post in blog_data:

    post.update(extract_interactions(post['content']))

    # Выделение жирным сущностей в выводе результатов

    post['markup'] = []

    for sentence_idx in range(len(post['sentences'])):

        s = post['sentences'][sentence_idx]
        for (term, _) in post['entity_interactions'][sentence_idx]:
            s = s.replace(term, '<strong>{0}</strong>'.format(term))

        post['markup'] += [s]

    filename = post['title'].replace("?", "") + '.entity_interactions.html'
    f = open(os.path.join(filename), 'wb')
    html = HTML_TEMPLATE.format(post['title'] + ' Interactions',
                                ' '.join(post['markup']))
    f.write(html.encode('utf-8'))
    f.close()

    print('Data written to', f.name)

    # Отображение любого из этих файлов в плавающем фрейме. Здесь отображается
    # последний обработанный файл, соответствующий последнему значению f.name

    print('Displaying {0}:'.format(f.name))
    display(IFrame('files/{0}'.format(f.name), '100%', '600px'))

```

Также было бы полезно провести дополнительный анализ для выявления наборов взаимодействий в более обширном тексте, отыскать и отобразить сопутствующие взаимодействия. Хорошей отправной точкой для такой визуализации мог бы стать код, использующий ориентированные графы сил, который представлен в примере 8.16. Но даже без знания конкретной природы взаимодействий информация о субъектах и объектах может очень пригодиться. Если вам интересен этот подход и вы достаточно честолобивы, попробуйте дополнить кортежи отсутствующими глаголами.

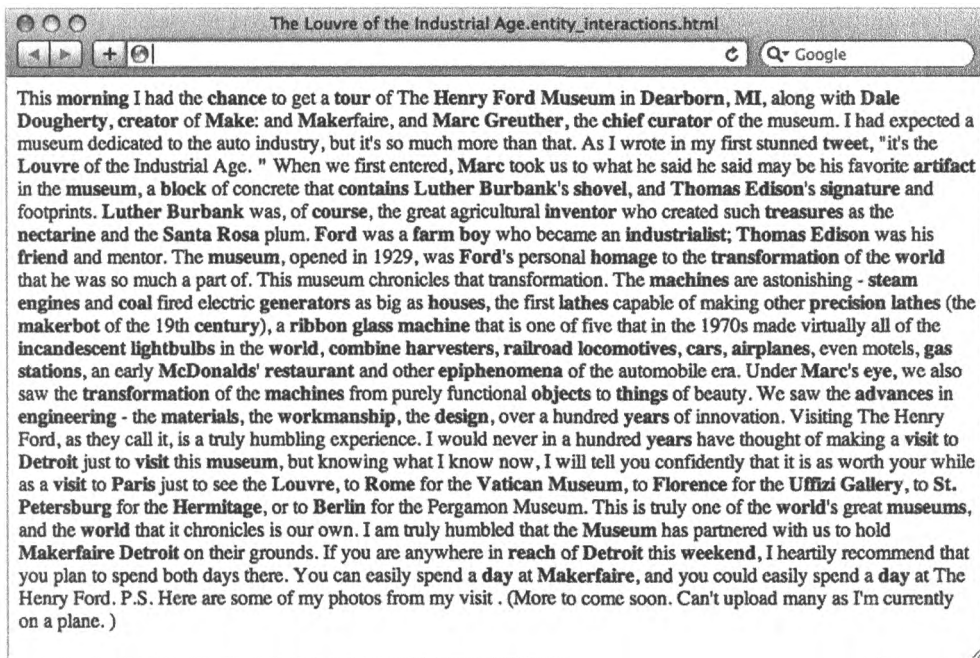


Рис. 6.6. Пример вывода текста в формате HTML, где сущности выделены жирным шрифтом, чтобы проще было увидеть ключевые понятия

6.5. Оценка качества при анализе данных на человеческом языке

После получения даже самого скромного объема результатов анализа текста неизбежно возникает желание оценить их качество. Насколько точно работает используемый алгоритм разделения предложений и/или маркировки частей речи? Например, как узнать, выполняя настройку базового алгоритма извлечения сущностей из неструктурированного текста, становится ли он более или менее эффективным в отношении качества получаемых результатов? Конечно, при работе с небольшим корпусом результаты можно проверять вручную и продолжать настраивать алгоритм, пока вас не удовлетворит его работа, но вам чертовски трудно будет выяснить, насколько хорошо ваш анализ справляется с большими корпусами или с документами другого типа. Следовательно, возникает необходимость автоматизировать процесс оценки.

Очевидной отправной точкой является случайная выборка некоторых документов и создание «золотого множества» сущностей, которые, по вашему мнению, обязательно должны извлекаться хорошим алгоритмом, и затем использовать этот список как основу для оценки. В зависимости от желаемой степени строгости можно даже вычислить ошибку выборки и использовать статистический аппарат с названием *доверительный интервал*¹ для предсказания действительной погрешности со степенью достоверности, удовлетворяющей вашим потребностям. Но как именно вычисляется оценка точности по результатам работы механизма извлечения и с использованием золотого множества? Часто для измерения точности используется так называемая *оценка F1*, которая определяется в терминах двух понятий — *точности и полноты*² (precision и recall):

$$F = 2 \times \frac{\text{точность} \times \text{полнота}}{\text{точность} + \text{полнота}},$$

где

$$\text{точность} = \frac{TP}{TP + FP},$$

а

$$\text{полнота} = \frac{TP}{TP + FN}.$$

В данном контексте под точностью подразумевается мера, отражающая ложноположительные срабатывания, а под полнотой — истинноположительные срабатывания. Далее приводится описание этих терминов для тех, кому они незнакомы или непонятны:

Истинноположительные (True Positives, TP)

Лексемы, которые правильно определены как сущности.

Ложноположительные (False Positives, FP)

Лексемы, которые ошибочно определены как сущности.

¹ <http://bit.ly/1a1n8BW>.

² Точнее говоря, F1 называется *гармоническим средним* точности и полноты, где гармоническое среднее любых двух чисел определяется как $H = 2 \times \frac{x \times y}{x + y}$.

Что такое «гармоническое среднее», вы сможете узнать в «Википедии», в статье с описанием *гармонических чисел* (<http://bit.ly/1a1n6tJ>).

Истинноотрицательные (True Negatives, TN)

Лексемы, которые правильно не были определены как сущности.

Ложноотрицательные (False Negatives, FN)

Лексемы, которые ошибочно не были определены как сущности.

Точность является мерой, учитывающей долю ложных срабатываний, поэтому она определяется как $TP / (TP + FP)$. Очевидно, что если доля ложных срабатываний равна нулю, тогда точность алгоритма является идеальной и оценка точности получает значение 1,0. Напротив, если число ложных срабатываний велико и приближается к числу истинных срабатываний или превосходит его, тогда оценка точности уменьшается и отношение приближается к нулю. Мера полноты определяется как $TP / (TP + FN)$, соответственно, если число ложных отрицаний равно нулю, она получает значение 1,0, что соответствует идеальной полноте. С увеличением числа ложных отрицаний оценка полноты приближается к нулю. По определению оценка F1 получает значение 1,0, когда точность и полнота идеальны, и приближается к нулю с ухудшением обоих показателей.

На практике же приходится искать компромисс, повышая точность или полноту, потому что трудно увеличивать и то и другое одновременно. Если задуматься, предыдущее утверждение не лишено смысла из-за взаимовлияния ложных срабатываний и ложных отрицаний (рис. 6.7).

Чтобы было понятнее, рассмотрим еще раз предложение «Mr. Green killed Colonel Mustard in the study with the candlestick» и предположим, что эксперт определил в нем следующие ключевые сущности: «Mr. Green», «Colonel Mustard», «study» и «candlestick». Допустим, ваш алгоритм выявил только эти четыре сущности, соответственно имели место четыре истинных срабатывания, ноль ложных срабатываний, пять истинных отрицаний («killed», «with», «the», «in», «the») и ноль ложных отрицаний. Это идеальные точность и полнота, которым соответствует оценка F1, равная 1,0. Подстановка разных значений в формулы оценки точности и полноты — простое и полезное упражнение для тех, кто впервые сталкивается с этими терминами.



Какие значения получили бы оценки точности, полноты и F1, если бы алгоритм выявил сущности «Mr. Green», «Colonel», «Mustard» и «candlestick»?

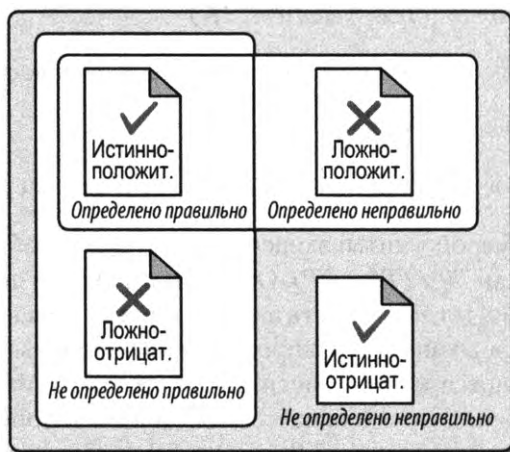


Рис. 6.7. Смысл понятий «истинное срабатывание», «ложное срабатывание», «истинное отрицание» и «ложное отрицание» с точки зрения прогнозного анализа

Многие из наиболее привлекательных стеков технологий, используемых коммерческими предприятиями в области NLP, включают обработку данных на естественном языке с применением продвинутых статистических моделей и алгоритмов обучения с учителем. Как рассказывалось выше в этой главе, обучение с учителем — это, по сути, подход, в котором алгоритму предлагаются выборки с исходными данными и ожидаемыми результатами, на основе которых тот обучает модель предсказывать кортежи с приемлемой точностью. Самое сложное в этом случае — получить обученную модель, которая достаточно хорошо обобщает входные данные, еще не встречавшиеся ей. Если модель показывает прекрасные результаты на обучающих данных, но плохие на данных, которые прежде ей не встречались, говорят, что она страдает проблемой *переобучения*. Для измерения качества модели часто используется методика, которая называется *перекрестной проверкой*. Согласно этой методике часть обучающих данных (например, одна треть) резервируется исключительно для целей тестирования модели, а для обучения используется только оставшаяся часть выборки.

6.6. Заключительные замечания

В этой главе были представлены основы анализа неструктурированных данных, а также показано, как использовать NLTK и собрать оставшуюся часть конвей-

ера NLP для извлечения сущностей из текста. Область изучения данных на человеческом языке находится на стыке невероятного количества дисциплин и пока все еще находится на стадии формирования. Несмотря на наши коллективные усилия и решение задачи NLP для большинства распространенных естественных языков, эту задачу смело можно назвать задачей века (по крайней мере, первой его половины).

Максимально используйте все возможности NLTK, а когда потребуется большая производительность или качество, засучите рукава и покопайтесь в академической литературе. Это, по общему признанию, очень сложная задача, но достойная приложения усилий, если вы намерены овладеть ею. В одной главе нельзя охватить все аспекты, но возможности огромны, в вашем распоряжении имеются прекрасные инструменты с открытым исходным кодом, и тех, кто решит овладеть наукой и искусством обработки данных на человеческом языке, ждет светлое будущее.



Исходный код примеров для этой и всех других глав доступен на GitHub¹ в удобном формате Jupyter Notebook, который вы можете опробовать, не покидая веб-браузера.

6.7. Упражнения

- Адаптируйте код примеров из этой главы, чтобы собрать несколько сотен статей или сообщений из интернета и обобщить их содержимое.
- Напишите веб-приложение, использующее набор инструментов, такой как Google App Engine, чтобы получить онлайн-инструмент для обобщения документов. (Учитывая, что недавно Yahoo! приобрела компанию Summly², специализировавшуюся на обобщении новостей для читателей, это упражнение должно стать для вас особенно вдохновляющим. По сообщениям, стоимость покупки составила около 30 миллионов долларов США.)
- Попробуйте с помощью инструментов стемминга из пакета NLTK получить кортежи (*сущность, основа предиката, сущность*), опираясь на код из примера 6.9.

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² <http://tcrn.ch/1a1n70L>.

- Познакомьтесь с инструментом WordNet¹, с которым вы обязательно столкнетесь рано или поздно, и попробуйте использовать его для определения предикатных фраз, которые встретятся вам при анализе данных на естественном языке.
- Попробуйте отобразить извлеченные сущности в виде облака тегов².
- Попробуйте реализовать свой механизм определения конца предложений как детерминированный парсер, основанный на логике, которую можно определять в виде правил в инструкциях if-then, и сравните его с механизмами в NLTK. Уместно ли пытаться моделировать язык с использованием детерминированных правил?
- Попробуйте реализовать обход небольшой выборки новостных статей или сообщений в блогах с помощью Scrapy и извлечь из них текст для обработки.
- Исследуйте байесовский классификатор³ в NLTK, реализующий обучение с учителем, который можно использовать для маркировки обучающих данных, например документов. Попробуйте обучить классификатор на документах с такими метками, как «спорт», «редакционная статья» и «прочее», организовав извлечение документов с помощью Scrapy. Оцените точность с использованием оценки F1.
- Возможны ли ситуации, когда гармоническое среднее точности и полноты является не лучшей оценкой? Когда может понадобиться увеличить точность в ущерб полноте? Когда может понадобиться увеличить полноту в ущерб точности?
- Попробуйте применить методы, описанные в этой главе, для анализа данных из Twitter. Используйте для маркировки частей речи такие замечательные библиотеки, как GATE Twitter⁴ и Carnegie Mellon Twitter NLP⁵.

¹ <http://bit.ly/1a1n7hj>.

² <http://bit.ly/1a1n5pO>.

³ <http://bit.ly/1a1n9Wt>.

⁴ <http://bit.ly/1a1nad2>.

⁵ <http://bit.ly/1a1n84Y>.

6.8. Онлайн-ресурсы

Ниже приводится список ссылок, упоминавшихся в этой главе, которые могут оказаться полезными для вас:

- Статья «The Automatic Creation of Literature Abstracts»¹.
- Модель «мешок слов»².
- Байесовский классификатор³.
- Библиотека `BeautifulSoup`⁴.
- Библиотека `boilerpipe`⁵.
- Статья «Boilerplate Detection Using Shallow Text Features»⁶.
- Поиск в ширину⁷.
- Поиск в глубину⁸.
- Библиотека Carnegie Mellon Twitter NLP⁹.
- Корпус Common Crawl¹⁰.
- Доверительный интервал¹¹.
- Репозиторий GitHub библиотеки d3-cloud¹².
- Библиотека GATE Twitter¹³.
- Онлайн-версия `boilerpipe`¹⁴.
- Стандарт HTML5¹⁵.

¹ <http://bit.ly/1a1n4Cj>.

² <http://bit.ly/1a1lHDF>.

³ <http://bit.ly/1a1n9Wt>.

⁴ <http://bit.ly/1a1mRit>.

⁵ <http://bit.ly/2sCIFET>.

⁶ <http://bit.ly/1a1mN21>.

⁷ <http://bit.ly/1a1mYdG>.

⁸ <http://bit.ly/1a1mVPd>.

⁹ <http://bit.ly/1a1n84Y>.

¹⁰ <http://amzn.to/1a1mXXb>.

¹¹ <http://bit.ly/1a1n8BW>.

¹² <http://bit.ly/1a1n5pO>.

¹³ <http://bit.ly/1a1nad2>.

¹⁴ <http://bit.ly/1a1mSTF>.

¹⁵ <http://bit.ly/1a1mRz5>.

- Микроданные, технология¹.
- Книга о NLTK².
- Проект Penn Treebank³.
- Scrapy⁴.
- Обучение с учителем⁵.
- Пул потоков выполнения⁶.
- Тест тьюринга⁷.
- Статья «Unsupervised Multilingual Sentence Boundary Detection»⁸.
- WordNet⁹.

¹ <http://bit.ly/1a1mRPA>.

² <http://bit.ly/1a1mtAk>.

³ <http://bit.ly/2C5ecDq>.

⁴ <http://bit.ly/1a1mG6P>.

⁵ <http://bit.ly/1a1mPHr>.

⁶ <http://bit.ly/1a1mW5M>.

⁷ <http://bit.ly/1a1mZON>.

⁸ <http://bit.ly/2EzWCEZ>.

⁹ <http://bit.ly/1a1n7hj>.

7

Анализ электронной почты: кто кому пишет, о чем, как часто и многое другое

Почтовые архивы являются, пожалуй, основным видом социальных данных и основой ранних социальных сетей. Электронная почта получила повсеместное распространение, и каждое сообщение по своей сути является социальным, будучи составной частью диалогов и взаимодействий между двумя и более людьми. Кроме того, каждое сообщение включает данные на человеческом языке, выразительные по своей природе, и пронизано полями структурированных метаданных, которые связывают данные на человеческом языке с определенными временными интервалами и обеспечивают их однозначную идентификацию. Анализ почтовых данных открывает возможность использовать все понятия, представленные в предыдущих главах, и получать новые ценные знания.

Если вы IT-директор некоей корпорации и желаете проанализировать тренды и шаблоны общения внутри своей организации, остро заинтересованы в исследовании списков почтовой рассылки или просто хотите изучить свой почтовый ящик на наличие закономерностей для собственной оценки¹, следующее пояснение покажет вам, с чего начать. Эта глава знакомит с некоторыми инструментами и приемами исследования почтовых ящиков, которые помогут ответить на такие вопросы, как:

- Кто и кому посылает письма (и как много/часто)?
- Есть ли какое-то определенное время суток (или дни недели), когда общение по электронной почте протекает особенно интенсивно?
- Кто и с кем переписывается чаще всего?
- На какие темы ведутся самые оживленные дискуссии?

¹ <http://bit.ly/1a1niJw>.

Сайты социальных сетей накапливают петабайты социальных данных почти в режиме реального времени, но их главный недостаток — централизованное управление социальными данными и наличие ограничивающих правил, которые регламентируют, как получить доступ к этим данным, а также что можно и чего нельзя с ними делать. Почтовые архивы, напротив, децентрализованы и разбросаны по всей сети в виде объемных списков рассылки с обсуждением множества тем, а также многих тысяч сообщений, спрятанных в учетных записях пользователей электронной почты. Представив себе все это, легко понять, что эффективный анализ почтовых архивов может оказаться одной из важнейших возможностей в вашем арсенале инструментов анализа данных.

Найти реалистичные наборы социальных данных для иллюстрации непросто, и тем не менее эта глава основывается на детально изученном корпусе Enron¹, использование которого помогает избежать юридических проблем² и проблем с неприкосновенностью частной переписки. Мы стандартизуем набор данных в широко известный формат почтового ящика UNIX (mbox), чтобы получить возможность использовать обширный набор инструментов для его обработки. Наконец, несмотря на возможность выбора формата JSON для представления данных и хранения их в одном плоском файле, мы воспользуемся мощной библиотекой `pandas` для анализа данных, представленной в главе 2, которая позволяет индексировать данные и выполнять запросы к ним.



Загрузите последний исправленный исходный код примеров для этой (и любой другой) главы, доступный по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Воспользуйтесь также преимуществами виртуальной машины, описанными в приложении А, чтобы получить максимальное удовольствие от опробования примеров кода.

¹ <http://bit.ly/1a1nj01>.

² Если вы захотите проанализировать данные из списка рассылки, имейте в виду, что большинство поставщиков услуг (таких, как Google и Yahoo!) ограничивают использование этих данных, если они извлекаются с помощью их программных интерфейсов, но вы легко сможете собрать свой архив почтовой рассылки, подписавшись на одну из них и подождя, пока ваш почтовый ящик наполнится. Как вариант, вы можете обратиться к владельцу или участникам рассылки и попросить их передать вам архив.

7.1. Обзор

Почтовые данные невероятно богаты и предоставляют самые широкие возможности для их анализа с привлечением всех средств и методов, с которыми вы познакомились ранее в этой книге. В этой главе вы познакомитесь с:

- особенностями обработки почтовых данных в стандартном, удобном и переносимом формате;
- библиотекой `pandas` для Python, содержащей мощные инструменты анализа табличных данных;
- корпусом Enron — общедоступным набором данных, содержащим письма из почтовых ящиков сотрудников в период скандала в компании Enron;
- приемами выполнения произвольных запросов `pandas` к корпусу Enron;
- инструментами для доступа к вашему собственному почтовому ящику и экспортирования его содержимого для дальнейшего анализа.

7.2. Получение и обработка корпуса с почтовыми сообщениями

В этом разделе вы узнаете, как получить корпус с почтовыми сообщениями, преобразовать его в стандартный формат `mbox` почтового ящика UNIX и затем импортировать их в экземпляр `DataFrame`, реализованный в библиотеке `pandas`, который играет роль универсального хранилища с поддержкой запросов. Сначала мы проанализируем содержимое небольшого вымышленного почтового ящика, а затем перейдем к обработке корпуса Enron.

7.2.1. Пример почтового ящика UNIX

На самом деле данные в формате `mbox` — это всего лишь очень большой текстовый файл с почтовыми сообщениями, которые можно извлекать с помощью любых инструментов, предназначенных для работы с текстом. Инструменты и протоколы электронной почты давно вышли за рамки формата `mbox`, но он часто используется в качестве наименьшего общего знаменателя, позволяя с легкостью обрабатывать данные и давая уверенность, что, если вы решите поделиться данными в этом формате, другие с такой же легкостью смогут об-

работать их. Многие почтовые клиенты предлагают возможность «экспортировать» или «сохранить как» для экспорта данных в этот формат (конкретное название пункта меню, отвечающего за эту операцию, может отличаться), как показано на рис. 7.5 в разделе «Анализ собственных почтовых данных».

Согласно спецификации формата mbox, каждое новое сообщение обозначается специальной строкой *From_*, соответствующей шаблону *From user@example.com asctime*, где *asctime* — это время в стандартном представлении *Fri Dec 25 00:06:42 2009*. Границы между сообщениями определяются по строке *From_*, которой предшествуют (кроме первого вхождения) точно два символа перевода строки. (Визуально, как показано в следующем фрагменте, они выглядят как одна пустая строка, предшествующая строке *From_*.) Вот фрагмент данных с двумя сообщениями в формате mbox из вымышленного почтового ящика:

```
From santa@northpole.example.org Fri Dec 25 00:06:42 2009
Message-ID: <16159836.1075855377439@mail.northpole.example.org>
References: <88364590.8837464573838@mail.northpole.example.org>
In-Reply-To: <194756537.0293874783209@mail.northpole.example.org>
Date: Fri, 25 Dec 2001 00:06:42 -0000 (GMT)
From: St. Nick <santa@northpole.example.org>
To: rudolph@northpole.example.org
Subject: RE: FWD: Tonight
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
```

```
Sounds good. See you at the usual location.
Thanks,
-S
```

```
-----Original Message-----
From: Rudolph
Sent: Friday, December 25, 2009 12:04 AM
To: Claus, Santa
Subject: FWD: Tonight
```

Santa -

Running a bit late. Will come grab you shortly. Stand by.

Rudy

Begin forwarded message:

```
> Last batch of toys was just loaded onto sleigh.
>
```

```
> Please proceed per the norm.  
>  
> Regards,  
> Buddy  
>  
> --  
> Buddy the Elf  
> Chief Elf  
> Workshop Operations  
> North Pole  
> buddy.the.elf@northpole.example.org
```

```
From buddy.the.elf@northpole.example.org Fri Dec 25 00:03:34 2009  
Message-ID: <88364590.8837464573838@mail.northpole.example.org>  
Date: Fri, 25 Dec 2001 00:03:34 -0000 (GMT)  
From: Buddy <buddy.the.elf@northpole.example.org>  
To: workshop@northpole.example.org  
Subject: Tonight  
Mime-Version: 1.0  
Content-Type: text/plain; charset=us-ascii  
Content-Transfer-Encoding: 7bit
```

Last batch of toys was just loaded onto sleigh.

Please proceed per the norm.

Regards,
Buddy

```
--  
Buddy the Elf  
Chief Elf  
Workshop Operations  
North Pole  
buddy.the.elf@northpole.example.org
```

В этом фрагменте мы видим два сообщения, хотя, судя по их содержимому, где-то в файле mbox существует еще одно сообщение. Первое в хронологическом порядке сообщение было написано человеком по имени Buddy, оно послано по адресу `workshop@northpole.example.org` и уведомляет, что только что были загружены игрушки. Другое сообщение в mbox — ответ человека по имени Santa человеку по имени Rudolph. В представленном фрагменте отсутствует промежуточное сообщение, в котором Rudolph переслал человеку по имени Buddy сообщение от Santa с пометкой о том, что он опаздывает. Мы, как люди, владеющие контекстом, могли бы определить все это сами, просто прочитав текст сообщений, однако в сообщениях есть заголовки **Message-ID**, **References**

и In-Reply-To, содержащие важные подсказки, которые можно было бы проанализировать.

Эти заголовки интуитивно понятны и представляют основу для алгоритмов, способных отобразить поток обсуждения. Чуть позже мы рассмотрим известный алгоритм, использующий эти поля для упорядочения сообщений, но пока важно понять, что каждое сообщение имеет уникальный идентификатор и идентификатор сообщения, на которое ссылается сообщение-ответ, если оно является ответом, с помощью которых можно построить цепочку ответов, являющихся частью всего потока обсуждения.



Поскольку некоторые рутинные задачи будут решаться с использованием модулей для Python, мы не будем отвлекаться на обсуждение нюансов электронной почты, таких как составное содержимое, типы MIME¹ и 7-битное кодирование для передачи.

Заголовки играют важную роль. Даже в этом простом примере уже можно видеть, насколько сложно порой выделить фактическое тело сообщения: почтовый клиент пользователя Rudolph отметил пересылаемое сообщение символами >, тогда как почтовый клиент пользователя Santa, пославший ответ, по всей видимости, исключил цитируемый текст, но добавил удобочитаемый заголовок.

Большинство почтовых клиентов отображают дополнительные заголовки электронной почты, кроме тех, что обычно можно наблюдать, и эта особенность может пригодиться, если вам интересен более доступный метод анализа, чем копание в исходном тексте в поисках информации, находящейся в этих заголовках. На рис. 7.1 показано, как отображаются заголовки в Apple Mail.

К счастью, многое возможно без необходимости писать собственный почтовый клиент. Кроме того, если вам нужно лишь просмотреть содержимое почтового ящика, достаточно импортировать его в почтовый клиент и просмотреть, верно?



Сейчас самое время выяснить, поддерживает ли ваш почтовый клиент импортирование/экспортирование данных в формате mbox, чтобы вы могли использовать инструменты из этой главы для управления ими.

¹ <http://bit.ly/1a1nmsJ>.

```

      From: Matthew Russell
      Subject: Message to self
      Date: September 28, 2010 9:31:01 PM CDT
      To: Matthew Russell
      Return-Path: <matthew@zafra.com>
      X-Spam-Checker-Version: SpamAssassin 3.1.9 (2007-02-13) on mail2.webfaction.com
      X-Spam-Level:
      X-Spam-Status: No, score=-2.6 required=5.0 tests=BAYES_00 autolearn=ham version=3.1.9
      Received: from smtp.webfaction.com (mail6.webfaction.com [74.55.86.74]) by
      mail2.webfaction.com (8.13.1/8.13.3) with ESMTP id o8T2V254026699 for
      <matthew@zafra.com>; Tue, 28 Sep 2010 21:31:02 -0500
      Received: from [192.168.1.67] (99-0-32-163.lightspeed.nsvln.sbcglobal.net
      [99.0.32.163]) by smtp.webfaction.com (Postfix) with ESMTP id
      9CE61324B7D for <matthew@zafra.com>; Tue, 28 Sep 2010 21:31:02 -0500
      (CDT)
      Message-Id: <D9A2277D-A6A1-4CD2-B891-C0A1E4C6C8CD@zafra.com>
      Content-Type: text/plain; charset=US-ASCII; format=flowed
      Content-Transfer-Encoding: 7bit
      Mime-Version: 1.0 (Apple Message framework v936)
      X-Mailer: Apple Mail (2.936)

Hello Matthew!

Regards - Matthew

---
http://www.linkedin.com/in/ptwobrussell

```

Рис. 7.1. Большинство почтовых клиентов поддерживают настройки, позволяющие видеть дополнительные заголовки

Для начала рассмотрим процедуру обработки данных, представленную в примере 7.1, которая использует множество упрощающих допущений о mbox и знакомит с пакетом `mailbox`, входящим в состав стандартной библиотеки Python.

Пример 7.1. Преобразование содержимого почтового ящика в формат JSON

```

import mailbox # pip install mailbox
import json

MBOX = 'resources/ch07-mailboxes/data/northpole.mbox'

# Следующая процедура использует множество упрощающих допущений
# для преобразования сообщения и mbox в объект на языке Python
# и на примере файла northpole.mbox демонстрирует
# основы парсинга формата mbox с помощью утилит для работы с почтой

def objectify_message(msg):

    # Словарь с полями из сообщения
    o_msg = dict([ (k, v) for (k,v) in msg.items() ])

    # Предполагается, что сообщение состоит из единственной части,

```

```

# и извлекается ее содержимое вместе с типом этого содержимого

part = [p for p in msg.walk()][0]
o_msg['contentType'] = part.get_content_type()
o_msg['content'] = part.get_payload()

return o_msg

# Загрузить сообщения из файла в формате mbox и последовательно
# преобразовать каждое из них в удобное представление JSON

mbox = mailbox.mbox(MBOX)

messages = []

for msg in mbox:
    messages.append(objectify_message(msg))

print(json.dumps(messages, indent=1))

```

Этот короткий сценарий, реализующий обработку файла mbox, выглядит довольно просто и выводит понятные результаты, однако вообще парсинг произвольных почтовых данных или определение точного потока обсуждения на их основе могут быть очень сложным предприятием. Этому способствуют многие факторы, такие как неоднозначность и возможные варианты вставки ответов и комментариев в цепочки обсуждения, различия в обработке сообщений и ответов разными почтовыми клиентами и т. д.

В табл. 7.1 показан поток сообщений, в который явно включено третье сообщение, на которое имелись ссылки, но оно отсутствовало в `northpole.mbox`. Далее приводится сокращенный вывод этого сценария:

```

[
{
  "From": "St. Nick <santa@northpole.example.org>",
  "Content-Transfer-Encoding": "7bit",
  "Content": "Sounds good. See you at the usual location.\n\nThanks...",
  "To": "rudolph@northpole.example.org",
  "References": "<88364590.8837464573838@mail.northpole.example.org>",
  "Mime-Version": "1.0",
  "In-Reply-To": "<194756537.0293874783209@mail.northpole.example.org>",
  "Date": "Fri, 25 Dec 2001 00:06:42 -0000 (GMT)",
  "contentType": "text/plain",
  "Message-ID": "<16159836.1075855377439@mail.northpole.example.org>",
  "Content-Type": "text/plain; charset=us-ascii",
  "Subject": "RE: FWD: Tonight"
},

```

```
{
  "From": "Buddy <buddy.the.elf@northpole.example.org>",
  "Subject": "Tonight",
  "Content-Transfer-Encoding": "7bit",
  "content": "Last batch of toys was just loaded onto sleigh. \n\nPlease...",
  "To": "workshop@northpole.example.org",
  "Date": "Fri, 25 Dec 2001 00:03:34 -0000 (GMT)",
  "contentType": "text/plain",
  "Message-ID": "<88364590.8837464573838@mail.northpole.example.org>",
  "Content-Type": "text/plain; charset=us-ascii",
  "Mime-Version": "1.0"
}
```

Таблица 7.1. Поток сообщений из northpole.mbox

Дата	Действия
Fri, 25 Dec 2001 00:03:34 -0000 (GMT)	Buddy послал сообщение в группу
Friday, December 25, 2009 12:04 AM	Rudolph переслал сообщение, полученное от Buddy, пользователю Santa с дополнительным замечанием
Fri, 25 Dec 2001 00:06:42 -0000 (GMT)	Пользователь Santa ответил пользователю Rudolph

Теперь, реализовав простейший анализ почтового ящика, перенесем наше внимание на преобразование корпуса Enron в формат mbox, чтобы максимально использовать стандартную библиотеку Python.

7.2.2. Получение корпуса Enron

Полный набор данных Enron¹ распространяется в нескольких форматах, требующих разного объема обработки. Мы начнем с данных в оригинальном исходном формате, которые организованы как множество папок с почтовыми ящиками — по одному на папку. Преобразование данных в стандартное представление и их очистка — во многом рутинная задача, и этот раздел поможет вам получить достаточно полное представление о том, как она решается.

Пользующиеся виртуальной машиной с примерами для этой книги найдут в блокноте Jupyter Notebook для этой главы сценарий, загружающий данные

¹ <http://bit.ly/1a1nmsU>.

в нужный каталог, чтобы дать возможность легко следовать за примерами. Полный корпус Enron имеет размер в сжатой форме около 450 Мбайт, и вам нужно его загрузить, чтобы опробовать все примеры в этой главе.

Первый этап обработки может занять продолжительное время. Если время является важным фактором и вы не можете позволить себе потратить его на выполнение сценария, просто пропустите этот этап; очищенные данные, возвращаемые примером 7.2, доступны в файле `ipynb3e/resources/ch07-mailboxes/data/enron.mbox.bz2`. Подробное описание, как его использовать, вы найдете в блокноте Jupyter Notebook для этой главы.



Загрузка и распаковка архива выполняются относительно быстро в сравнении с операцией синхронизации большого числа файлов, которые распаковываются хост-машиной, и на момент написания этих строк не было обходного решения, с помощью которого можно было бы ускорить этот процесс на всех платформах.

Следующий листинг сеанса в терминале с комментариями иллюстрирует базовую структуру корпуса после загрузки и распаковки.



Если вы пользуетесь операционной системой Windows или вам некомфортно работать в терминале, покопайтесь в папке `ipynb/resources/ch06-mailboxes/data`, синхронизированной с хост-машиной, при условии что вы пользуетесь виртуальной машиной для опробования примеров.

Загрузив данные, потратьте несколько минут на их исследование в терминале, чтобы узнать, как они организованы и что в них имеется:

```
$ cd enron_mail_20110402/maildir # Перейти в каталог с почтой
```

```
maildir $ ls # Вывести список папок/файлов в текущем каталоге
```

allen-p	crandell-s	gay-r	horton-s
lokey-t	nemec-g	rogers-b	slinger-r
tycholiz-b	arnold-j	cuilla-m	geaccone-t
hyatt-k	love-p	panus-s	ruscitti-k
smith-m	ward-k	arora-h	dasovich-j
germany-c	hyvl-d	lucci-p	parks-j
sager-e	solberg-g	watson-k	badeer-r
corman-s	gang-l	holst-k	lokay-m

...список каталогов усечен...

neal-s	rodrique-r	skilling-j	townsend-j
--------	------------	------------	------------

```
$ cd allen-p/ # Перейти в папку allen-p

allen-p $ ls # Вывести список файлов в текущем каталоге

_sent_mail      contacts          discussion_threads notes_inbox
sent_items      all_documents    deleted_items      inbox
sent            straw

allen-p $ cd inbox/ # Перейти в папку inbox в каталоге allen-p

inbox $ ls # Вывести список файлов в каталоге inbox для allen-p

1. 11. 13. 15. 17. 19. 20. 22. 24. 26. 28. 3. 31. 33. 35. 37. 39. 40.
42. 44. 5. 62. 64. 66. 68. 7. 71. 73. 75. 79. 83. 85. 87. 10. 12. 14.
16. 18. 2. 21. 23. 25. 27. 29. 30. 32. 34. 36. 38. 4. 41. 43. 45. 6.
63. 65. 67. 69. 70. 72. 74. 78. 8. 84. 86. 9.

inbox $ head -20 1. # Вывести первые 20 строк из файла с именем "1."

Message-ID: <16159836.1075855377439.JavaMail.evans@thyme>
Date: Fri, 7 Dec 2001 10:06:42 -0800 (PST)
From: heather.dunton@enron.com
To: k..allen@enron.com
Subject: RE: West Position
Mime-Version: 1.0
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit
X-From: Dunton, Heather </O=ENRON/OU=NA/CN=RECIPIENTS/CN=HDUNTON>
X-To: Allen, Phillip K. </O=ENRON/OU=NA/CN=RECIPIENTS/CN=Pallen>
X-cc:
X-bcc:
X-Folder: \Phillip_Allen_Jan2002_1\Allen, Phillip K.\Inbox
X-Origin: Allen-P
X-FileName: pallen (Non-Privileged).pst

Please let me know if you still need Curve Shift.

Thanks,
```

Заключительные команды в этом сеансе показывают, что сообщения организованы в файлы и содержат метаданные в виде заголовков, доступных для обработки наряду с содержимым сообщений. Данные имеют вполне согласованный формат, но не всегда поддерживаемый инструментами обработки. Поэтому выполним кое-какую предварительную обработку и преобразуем часть данных в формат mbox почтовых ящиков UNIX, чтобы познакомиться с типичным процессом стандартизации почтового корпуса в хорошо известный и широко поддерживаемый формат.

7.2.3. Преобразование почтового корпуса в формат mbox

Пример 7.2 иллюстрирует подход, который предусматривает поиск в дереве каталогов корпуса Enron папок с именами «inbox» и добавление сообщений из них в один выходной файл с именем `enron.mbox`. Чтобы запустить этот сценарий, необходимо прежде загрузить корпус Enron и распаковать его в каталог, указанный в переменной `MAILDIR` внутри сценария.

Для преобразования дат в стандартный формат сценарий использует пакет `dateutil`. Мы не выполняли этого преобразования ранее, и оно может оказаться сложнее, чем кажется на первый взгляд, учитывая большое разнообразие представлений дат в общем случае. Установить этот пакет можно командой `pip install python_dateutil`. (В данном случае имя пакета, устанавливаемого с помощью команды `pip`, отличается от имени, под которым пакет импортируется в коде.) В отсутствие этого пакета для преобразования дат в формат mbox сценарий будет использовать некоторые другие инструменты из стандартной библиотеки Python. С точки зрения анализа данных сценарий не представляет особого интереса, но он напоминает, как пользоваться регулярными выражениями, демонстрирует применение пакета `email`, который мы продолжим использовать в последующих примерах, и иллюстрирует некоторые другие понятия, характерные для обработки данных вообще. Обязательно разберитесь, как работает пример 7.2, чтобы расширить ваши общие познания в области обработки данных и применяемых инструментов.



Для обработки всего корпуса Enron этому сценарию может потребоваться 10–15 минут, в зависимости от быстродействия компьютера. На протяжении всего периода обработки Jupyter Notebook будет отображать в правом верхнем углу сообщение «Kernel Busy».

Пример 7.2. Преобразование корпуса Enron в стандартный формат mbox

```
import re
import email
from time import asctime
import os
import sys
from dateutil.parser import parse # pip install python_dateutil

# Загрузите корпус и распакуйте в папку resources/ch07-mailboxes/data

MAILDIR = 'resources/ch07-mailboxes/data/enron_mail_20110402/maildir'

# Файл для записи результатов преобразования
```

```
MBOX = 'resources/ch07-mailboxes/data/enron.mbox'

# Создать дескриптор выходного файла
mbox = open(MBOX, 'w+')

# Выполнить обход каталогов и обработать все папки 'inbox'
for (root, dirs, file_names) in os.walk(MAILDIR):
    if root.split(os.sep)[-1].lower() != 'inbox':
        continue

    # Обработать каждое сообщение в папке 'inbox'
    for file_name in file_names:

        file_path = os.path.join(root, file_name)
        message_text = open(file_path, errors='ignore').read()

        # Определить поля для строки From_ в типичном сообщении mbox
        _from = re.search(r"From: ([^\r\n]+)", message_text).groups()[0]
        _date = re.search(r>Date: ([^\r\n]+)", message_text).groups()[0]

        # Преобразовать _date в представление asctime для строки From_
        _date = asctime(parse(_date).timetuple())

        msg = email.message_from_string(message_text)
        msg.set_unixfrom('From {0} {1}'.format(_from, _date))

        mbox.write(msg.as_string(unixfrom=True) + "\n\n")

mbox.close()
```

Заглянув в только что созданный mbox-файл, вы увидите, что он выглядит очень похоже на почтовый формат, показанный выше, но теперь точно соответствует спецификации, и все сообщения хранятся в одном файле.



При необходимости точно так же легко можно создать по отдельному файлу для каждого пользователя или группы пользователей, если вдруг вам потребуется проанализировать ограниченное подмножество корпуса Enron.

7.2.4. Преобразование почтовых ящиков UNIX в объекты DataFrames

Формат mbox удобен еще и тем, что для работы с ним имеется множество инструментов, написанных на разных языках программирования и действующих

щих на разных платформах. В этом разделе мы устраним многие упрощающие допущения, принятые в примере 7.1, чтобы получить возможность надежно обрабатывать содержимое почтового корпуса Enron и разрешить некоторые из распространенных проблем, с которыми вам наверняка придется столкнуться на практике при работе с почтовыми данными.

Структура данных `mbox` универсальна, но чтобы получить широкие возможности для манипуляции данными, выполнения запросов и визуализации, мы преобразуем ее в структуру данных `DataFrame`, представленную в главе 2 и реализованную в библиотеке `pandas`.

Напомню, что `pandas` — это программная библиотека, написанная на Python, которая предлагает универсальные средства для анализа данных. Она достойна стать частью арсенала любого специалиста, занимающегося анализом данных. В библиотеке `pandas` реализована структура данных `DataFrame`, поддерживающая возможность хранения данных в виде двумерных таблиц с именованными столбцами. Их можно рассматривать как электронные таблицы, в которых каждый столбец имеет имя. Столбцы в одной таблице могут иметь разные типы данных. Например, один столбец в таблице может хранить даты, другой — строки, а третий — вещественные числа, и все это в одной структуре `DataFrame`.

Сообщения электронной почты прекрасно подходят для хранения в `DataFrame`. В одном столбце можно хранить поле *From*, в другом — строку темы письма, и т. д. После записи и индексирования данных в `DataFrame` появляется возможность выполнять запросы («какие письма содержат данное слово?») и вычислять статистики («сколько писем было отправлено в апреле?»). Библиотека `pandas` значительно упрощает выполнение всех этих операций.

Код в примере 7.3 сначала преобразует корпус Enron из формата `mbox` в формат словаря Python. Переменная `mbox_dict` хранит все электронные письма как отдельные пары ключ/значение, причем каждое письмо также структурировано в виде словаря Python, ключами которого служат названия заголовков (*To*, *From*, *Subject* и т. д.) и само содержимое письма. Полученный словарь легко можно преобразовать в `DataFrame` вызовом метода `from_dict`.

Пример 7.3. Преобразование данных из формата `mbox` в словарь Python и далее в структуру `DataFrame`

```
import pandas as pd # pip install pandas
import mailbox

MBOX = 'resources/ch07-mailboxes/data/enron.mbox'
```

```

mbox = mailbox.mbox(MBOX)

mbox_dict = {}
for i, msg in enumerate(mbox):
    mbox_dict[i] = {}
    for header in msg.keys():
        mbox_dict[i][header] = msg[header]
    mbox_dict[i]['Body'] = msg.get_payload().replace('\n', ' ')
        .replace('\t', ' ').replace('\r', ' ').strip()

df = pd.DataFrame.from_dict(mbox_dict, orient='index')

```

Первые пять записей в DataFrame можно вывести с помощью метода `head`. На рис. 7.2 показано, как выглядит вывод этой команды.

In [13]: `df.head()`

Out[13]:

	Message-ID	Date	From	To	Subject	Mime-Version	Content-Type	Content-Transfer-Encoding
0	<10682204.1075852727740.JavaMail.evans@thyme>	Thu, 26 Oct 2001 09:16:59 -0700 (PDT)	kimberly.banner@enron.com	john.arnold@enron.com	hi	1.0	text/plain; charset=us-ascii	7bit
1	<14163285.1075861678333.JavaMail.evans@thyme>	Tue, 27 Nov 2001 10:25:00 -0800 (PST)	kbusch@energyargus.com	john.arnold@enron.com	keeping the lights on	1.0	text/plain; charset=us-ascii	7bit
2	<4724114.1075855217865.JavaMail.evans@thyme>	Mon, 31 Dec 2001 13:20:07 -0800 (PST)	customerservice@qwikdiks.com	john.arnold@enron.com	L.A. Confidential has been received.	1.0	text/plain; charset=us-ascii	7bit
3	<22448070.1075861675831.JavaMail.evans@thyme>	Tue, 20 Nov 2001 06:57:42 -0800 (PST)	tanya.rohauer@enron.com	john.arnold@enron.com	AIG positions	1.0	text/plain; charset=us-ascii	7bit
4	<29857442.1075855217869.JavaMail.evans@thyme>	Tue, 1 Jan 2002 11:04:18 -0800 (PST)	msagel@home.com	jaarnold@enron.com	Market update	1.0	text/plain; charset=us-ascii	7bit

5 rows x 24 columns

Рис. 7.2. Метод `head` класса DataFrame выводит первые пять записей из структуры DataFrame — это очень удобный способ быстрой проверки имеющихся данных

Заголовки письма содержат массу информации. Теперь каждый заголовок хранится в отдельном столбце в DataFrame. Чтобы получить список имен столбцов, можно воспользоваться командой `df.columns`.

Не все эти столбцы представляют интерес для нас, поэтому выберем некоторые из них и сделаем таблицу DataFrame немного компактнее. Мы оставим только поля *From*, *To*, *Cc*, *Bcc*, *Subject* и *Body*.

Еще одно преимущество структуры DataFrame — возможность выбора способа индексирования. Выбор индекса сильно влияет на скорость выполнения запросов. Если предполагается, что большая часть запросов будет проверять время отправки писем, *тогда* имеет смысл установить в DataFrame индекс `DatetimeIndex`. В результате данные в DataFrame будут отсортированы по дате и времени, что позволит выполнять быстрый поиск писем по времени отправки. Код в примере 7.4 показывает, как установить индекс в DataFrame и как выбрать подмножество столбцов для сохранения.

Пример 7.4. Установка индекса и выбор подмножества столбцов в DataFrame

```
df.index = df['Date'].apply(pd.to_datetime)

# Удалить ненужные столбцы
cols_to_keep = ['From', 'To', 'Cc', 'Bcc', 'Subject', 'Body']
df = df[cols_to_keep]
```

Теперь метод `head` вернет результаты, изображенные на рис. 7.3.

In [15]: df.head

Out[15]:

Date	From	To	Cc	Bcc	Subject	Body
2001-10-25 16:18:50	kimberly.banner@enron.com	john.arnold@enron.com		NaN	NaN	Hi Hey there. We are going to be at Front Porch ...
2001-11-27 18:25:00	kbusch@energyargus.com	john.arnold@enron.com		NaN	NaN	keeping the lights on John- Was lovely meeting you this weekend. So...
2001-12-31 21:20:07	customerservice@qwiklist.com	john.arnold@enron.com		NaN	NaN	L.A. Confidential has been received. [IMAGE] Qwiklist() MyAccount Rental Histor...
2001-11-30 14:57:42	tanya.rohauer@enron.com	john.arnold@enron.com	joey.taylor@enron.com	joey.taylor@enron.com	AIG positions	John, As discussed, the AIG exposure is \$57MM...
2002-01-01 18:04:16	msage@home.com	jarold@enron.com	mmaggi@enron.com	mmaggi@enron.com	Market update	Happy New Year and best wishes. - Energy01010...

Рис. 7.3. Вывод метода `head` после применения изменений, реализованных в примере 7.4

Теперь, преобразовав данные из формата `mbox` в формат `DataFrame` и выбрав интересные нас столбцы, мы можем приступить к анализу.

7.3. Анализ корпуса Enron

Уделив немалое внимание проблеме преобразования данных Enron в удобный формат с поддержкой запросов, мы готовы продолжить наш путь и заняться анализом данных. Как не раз упоминалось в предыдущих главах, столкнув-

шись с новыми данными, исследователи обычно начинают их анализ с подсчета, потому что количественные характеристики позволяют узнать многое с минимальными усилиями. В этом разделе, в порядке рабочего обсуждения, описывается несколько приемов использования библиотеки *pandas* для выполнения запросов с разными комбинациями полей и критериев, требующих минимальных усилий.

ДЕЛО ENRON

Хотя в этом нет необходимости, но, как нам кажется, вам будет полезно узнать о скандале с компанией Enron, который стал предметом почтовых данных, предлагаемых для анализа. Ниже приводится несколько основных фактов о компании Enron, которые помогут лучше понять контекст в процессе анализа данных в этой главе:

- Enron — энергетическая компания в штате Техас, которая выросла в многомиллиардную компанию с момента ее основания в 1985-го и до скандала, разразившегося в октябре 2001-го.
- Кеннет Лей (Kenneth Lay) — президент Enron и объект многих дискуссий, касающихся Enron.
- Суть скандала с компанией Enron заключалась в использовании финансовых инструментов (названных *хищниками*, или *рапторами* — *raptors*¹) для сокрытия убытков.
- Arthur Andersen — некогда престижная бухгалтерская фирма, отвечавшая за проведение финансового аудита. Закрылась вскоре после скандала с Enron.
- Вскоре после разразившегося скандала Enron объявила о банкротстве на сумму более 60 миллиардов долларов США; это было крупнейшее банкротство за всю историю США.

Статья² в «Википедии», посвященная скандалу Enron, в простой и доступной форме перечисляет основные события, и вам будет достаточно нескольких минут, чтобы понять суть происшедшего. Желющие узнать больше могут посмотреть документальный фильм *Enron: The Smartest Guys in the Room*³ (*Enron. Самые смышленные парни в этой комнате*).



На веб-сайте <http://www.enron-mail.com> размещена версия почтовых данных компании Enron, которые могут быть полезны для первого знакомства с корпусом Enron.

¹ В действительности — дочерние компании, создаваемые топ-менеджментом компании Enron и получавшие «хищные» названия, такие как «Raptor I», «Talon». — *Примеч. пер.*

² <http://bit.ly/1a1nuZo>.

³ <http://imdb.to/1a1nvwd>.

7.3.1. Запрос по диапазону времени

В примере 7.4 мы заменили в `DataFrame` индекс, соответствующий простому номеру записи, индексом по дате и времени. Это дает возможность быстро извлекать записи по времени отправки писем и отвечать на такие вопросы, как:

- Сколько писем было отправлено 1 ноября 2001 года?
- В каком месяце было отправлено больше всего писем?
- Сколько всего писем в неделю отправлялось в 2002 году?

В `pandas` имеется несколько удобных функций индексирования¹, наиболее важными из которых являются методы `loc` и `iloc`. Метод `loc` главным образом ориентирован на выбор по меткам и позволяет выполнять такие запросы, как:

```
df.loc[df.From == 'kenneth.lay@enron.com']
```

Эта команда вернет все письма, отправленные Кеннетом Леем (Kenneth Lay) и присутствующие в `DataFrame`. Результатом является такая же структура `DataFrame`, то есть ее можно присвоить переменной и использовать для других целей.

Метод `iloc` ориентирован на выбор по позиции, что очень удобно, когда заранее точно известны номера интересующих вас записей. Он позволяет выполнять такие запросы, как:

```
df.iloc[10:15]
```

Этот вызов вернет срез набора данных, содержащий записи с 10 по 15.

А теперь представьте, что нам нужно получить все письма, отправленные или полученные в период между двумя конкретными датами. Это тот случай, когда нам очень пригодится наш `DatetimeIndex`. Как показано в примере 7.5, мы легко можем извлечь все письма, отправленные в указанный период, и затем выполнить с полученным срезом желаемые операции, например, подсчитать число писем, отправленных за месяц.

Пример 7.5. Извлечение выборки за определенный период с использованием `DatetimeIndex` и подсчет числа писем, отправленных в течение месяца

```
start_date = '2000-1-1'  
stop_date = '2003-1-1'
```

¹ <http://bit.ly/2HGyotX>.

```
datemask = (df.index > start_date) & (df.index <= stop_date)
vol_by_month = df.loc[datemask].resample('1M').count()['To']

print(vol_by_month)
```

Здесь мы создали логическую маску, которая возвращает True, если запись в DataFrame удовлетворяет двойному ограничению — дата отправки позже start_date, но раньше end_date. Затем производится выборка и подсчет записей за каждый месяц. Вот как выглядит вывод примера 7.5:

```
Date
2000-12-31      1
2001-01-31      3
2001-02-28      2
2001-03-31     21
2001-04-30    720
2001-05-31   1816
2001-06-30   1423
2001-07-31    704
2001-08-31   1333
2001-09-30   2897
2001-10-31   9137
2001-11-30   8569
2001-12-31   4167
2002-01-31   3464
2002-02-28   1897
2002-03-31    497
2002-04-30     88
2002-05-31     82
2002-06-30    158
2002-07-31     0
2002-08-31     0
2002-09-30     0
2002-10-31     1
2002-11-30     0
2002-12-31     1
Freq: M, Name: To, dtype: int64
```

Теперь приведем результаты в более удобочитаемый вид. Мы можем использовать prettytable, как уже делали раньше, чтобы получить текстовую таблицу с результатами. Как это делается, показано в примере 7.6.

Пример 7.6. Вывод результатов подсчета писем по месяцам с помощью prettytable

```
from prettytable import PrettyTable

pt = PrettyTable(field_names=['Year', 'Month', 'Num Msgs'])
```

```
pt.align['Num Msgs'], pt.align['Month'] = 'r', 'r'
[ pt.add_row([ind.year, ind.month, vol])
  for ind, vol in zip(vol_by_month.index, vol_by_month)]

print(pt)
```

Этот код выведет такую таблицу:

Year	Month	Num Msgs
2000	12	1
2001	1	3
2001	2	2
2001	3	21
2001	4	720
2001	5	1816
2001	6	1423
2001	7	704
2001	8	1333
2001	9	2897
2001	10	9137
2001	11	8569
2001	12	4167
2002	1	3464
2002	2	1897
2002	3	497
2002	4	88
2002	5	82
2002	6	158
2002	7	0
2002	8	0
2002	9	0
2002	10	1
2002	11	0
2002	12	1

Конечно, еще лучше было бы отобразить эту таблицу в виде графика, что можно быстро реализовать с помощью функций построения графиков, уже имеющих-ся в библиотеке `pandas`. В примере 7.7 показано, как создать горизонтальную столбиковую диаграмму с помесечным количеством писем.

Пример 7.7. Вывод результатов подсчета писем по месяцам в виде горизонтальной столбиковой диаграммы

```
vol_by_month[::-1].plot(kind='barh', figsize=(5,8), title='Email Volume by Month')
```

Результат работы этого кода показан на рис. 7.4.

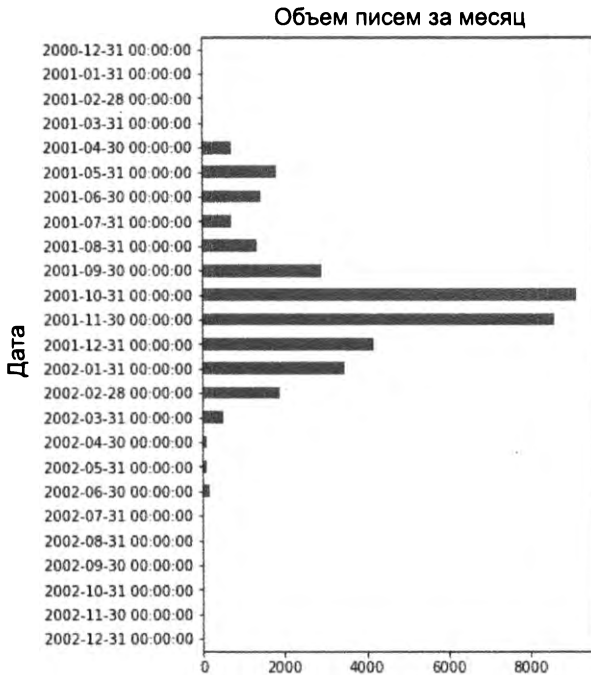


Рис. 7.4. Горизонтальная столбиковая диаграмма с результатами подсчета писем по месяцам, полученная в примере 7.7



Самый простой способ последовать за этими примерами — использовать самый свежий код, доступный по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Открыв этот код в среде Jupyter Notebook, вы легко сможете опробовать разные запросы к данным в DataFrame. Jupyter Notebook — замечательный инструмент для исследования данных.

7.3.2. Анализ закономерностей во взаимодействиях отправителей и получателей

Другие показатели, такие как количество писем, изначально созданных данным человеком, или количество прямых взаимодействий в некоторой группе лиц, тоже являются весьма представительными статистиками, которые желательно учитывать при анализе электронной почты. Однако прежде чем выяснять, кто

с кем общается, можно просто перечислить всех отправителей и получателей, при необходимости ограничив запрос критерием, таким как имя домена, откуда отправлено письмо или куда оно было доставлено. Для начала попробуем подсчитать количество адресов электронной почты отправителей или получателей, как показано в примере 7.8.

Пример 7.8. Перечисление отправителей и получателей сообщений

```
senders = df['From'].unique()
receivers = df['To'].unique()
cc_receivers = df['Cc'].unique()
bcc_receivers = df['Bcc'].unique()

print('Num Senders:', len(senders))
print('Num Receivers:', len(receivers))
print('Num CC Receivers:', len(cc_receivers))
print('Num BCC Receivers:', len(bcc_receivers))
```

Пример 7.8 выведет следующие сведения:

```
Num Senders: 7678
Num Receivers: 10556
Num CC Receivers: 5449
Num BCC Receivers: 5449
```

Даже без всякой дополнительной информации эти числа уже представляют большой интерес. В среднем каждое письмо отправлялось 1,4 адресатам, при этом также отправлялось значительное число копий (СС) и скрытых копий (ВСС). На следующем шаге можно попробовать добавить критерии выборки и применить к данным основные операции с множествами¹ (как было показано в главе 1), чтобы определить, как перекрываются разные комбинации этих критериев. Для этого достаточно просто преобразовать списки, содержащие каждое уникальное значение, в множества и выполнить такие операции сравнения множеств², как пересечение, разность и объединение. Для демонстрации в табл. 7.2 перечислены эти основные операции над следующей небольшой совокупностью отправителей и получателей:

```
Senders = {Abe, Bob}, Receivers = {Bob, Carol}
```

Пример 7.9 демонстрирует, как выполнять операции с множествами в коде на Python.

¹ <http://bit.ly/1a1l2Sw>.

² <http://bit.ly/2IzBW2j>.

Таблица 7.2. Пример операций с множествами

Операция	Название операции	Результат	Комментарий
$\text{Senders} \cup \text{Receivers}$	Объединение	Abe, Bob, Carol	Все уникальные отправители и получатели
$\text{Senders} \cap \text{Receivers}$	Пересечение	Bob	Отправители, которые также получали письма
$\text{Senders} - \text{Receivers}$	Разность	Abe	Отправители, не получавшие писем
$\text{Receivers} - \text{Senders}$	Разность	Carol	Получатели, не посылавшие писем

Пример 7.9. Определение числа отправителей и получателей в корпусе Enron

```

senders = set(senders)
receivers = set(receivers)
cc_receivers = set(cc_receivers)
bcc_receivers = set(bcc_receivers)

# Найти число отправителей, которые также являются прямыми получателями

senders_intersect_receivers = senders.intersection(receivers)

# Найти число отправителей, которые не получали никаких писем

senders_diff_receivers = senders.difference(receivers)

# Найти число получателей, которые не посылали никаких писем

receivers_diff_senders = receivers.difference(senders)

# Найти число отправителей, которые были получателями отправлений любых видов,
# предварительно найдя объединение всех видов получателей

all_receivers = receivers.union(cc_receivers, bcc_receivers)
senders_all_receivers = senders.intersection(all_receivers)

print("Num senders in common with receivers:", len(senders_intersect_receivers))
print("Num senders who didn't receive:", len(senders_diff_receivers))
print("Num receivers who didn't send:", len(receivers_diff_senders))
print("Num senders in common with *all* receivers:", len(senders_all_receivers))

```

Результаты работы этого сценария помогают получить некоторые дополнительные сведения о природе почтовых данных:

```

Num senders in common with receivers: 3220
Num senders who didn't receive: 4445
Num receivers who didn't send: 18942
Num senders in common with all receivers: 3440

```

Другой вопрос, ответ на который может представлять интерес: кто отправил или получил наибольшее число писем? Допустим, нам понадобилось составить рейтинги отправителей и получателей электронных писем в корпусе. Как бы мы решили эту задачу? Для этого можно было бы сгруппировать набор данных по отправителю или получателю, а затем подсчитать число писем в каждой группе. Каждая группа в этом случае будет представлять письма, отправленные одним отправителем или полученные одним получателем, и на их основе можно создать сортированные списки отправителей и получателей.

Такая операция группировки — частое явление в реляционных базах данных, и она также реализована в `pandas`. В примере 7.10 для такой группировки используется операция `groupby`¹.

Пример 7.10. Получение списков рейтингов отправителей и получателей в корпусе Enron

```
import numpy as np

top_senders = df.groupby('From')
top_receivers = df.groupby('To')

top_senders = top_senders.count()['To']
top_receivers = top_receivers.count()['From']

# Получить индексы отправителей и получателей в порядке убывания
top_snd_ord = np.argsort(top_senders[::-1])
top_rcv_ord = np.argsort(top_receivers[::-1])

top_senders = top_senders[top_snd_ord]
top_receivers = top_receivers[top_rcv_ord]
```

Создав сортированные списки, мы можем вывести их с помощью пакета `prettytable`. Пример 7.11 демонстрирует, как получить текстовую таблицу с 10 самыми активными отправителями.

Пример 7.11. Поиск 10 самых активных отправителей в корпусе Enron

```
from prettytable import PrettyTable

top10 = top_senders[:10]
pt = PrettyTable(field_names=['Rank', 'Sender', 'Messages Sent'])
pt.align['Messages Sent'] = 'r'
[ pt.add_row([i+1, email, vol]) for i, email, vol in zip(range(10),
    top10.index.values, top10.values)]

print(pt)
```

¹ <http://bit.ly/2FRGnno>.

Этот код выведет такую таблицу:

Rank	Sender	Messages Sent
1	pete.davis@enron.com	722
2	announcements.enron@enron.com	372
3	jae.black@enron.com	322
4	enron_update@concureworkplace.com	213
5	feedback@intcx.com	209
6	chairman.ken@enron.com	197
7	arsystem@mailman.enron.com	192
8	mike.grigsby@enron.com	191
9	soblender@carrfut.com	186
10	mary.cook@enron.com	186

Аналогично можно вывести таблицу для получателей. Как это сделать, демонстрирует пример 7.12.

Пример 7.12. Поиск 10 самых активных получателей в корпусе Enron

```
from prettytable import PrettyTable

top10 = top_receivers[:10]
pt = PrettyTable(field_names=['Rank', 'Receiver', 'Messages Received'])
pt.align['Messages Received'] = 'r'
[ pt.add_row([i+1, email, vol]) for i, email, vol in zip(range(10),
    top10.index.values, top10.values)]

print(pt)
```

Этот код выведет следующую таблицу:

Rank	Receiver	Messages Received
1	pete.davis@enron.com	721
2	gerald.nemec@enron.com	677
3	kenneth.lay@enron.com	608
4	sara.shackleton@enron.com	453
5	jeff.skilling@enron.com	420
6	center.dl-portland@enron.com	394
7	jeff.dasovich@enron.com	346
8	tana.jones@enron.com	303
9	rick.buy@enron.com	286
10	barry.tycholiz@enron.com	280

7.3.3. Поиск писем по ключевым словам

Обширные возможности индексирования¹ библиотеки `pandas` можно также использовать для выполнения поисковых запросов.

Например, чтобы в формате `mbox` отыскать конкретный адрес электронной почты, можно сначала проверить заголовки *To* и *From*, а затем исследовать заголовки *Cc* и *Bcc*. А для поиска по ключевым словам можно выполнить сопоставление с искомыми строками полей, содержащих тему и тело письма.

В контексте Enron *хищниками*² были финансовые махинации, которые привели к многомиллионным убыткам. Если бы мы проводили аудит, нам мог бы понадобиться инструмент для быстрого поиска во всем массиве электронных писем или документов. Мы уже подготовили корпус, преобразовав его в формат (`DataFrame`), упрощающий анализ данных в среде Python. Теперь нам просто нужно научиться выполнять поиск определенных слов, таких как «*raptor*». Один из таких способов демонстрирует код в примере 7.13.

Пример 7.13. Выполнение запросов в `DataFrame` с целью поиска ключевого слова в строке с темой и в теле письма и вывод первых 10 результатов

```
import textwrap

search_term = 'raptor'

query = (df['Body'].str.contains(search_term, case=False) |
        df['Subject'].str.contains(search_term, case=False))

results = df[query]

print('{0} results found.'.format(query.sum()))
print('Printing first 10 results...')
for i in range(10):
    subject, body = results.iloc[i]['Subject'], results.iloc[i]['Body']
    print()
    print('SUBJECT: ', subject)
    print('-'*20)
    for line in textwrap.wrap(body, width=70, max_lines=5):
        print(line)
```

В примере 7.13 мы определили логическое выражение и сохранили его в переменной `query`. Этот запрос проверяет текст и темы электронных писем в нашем

¹ <http://bit.ly/2HGyotX>.

² <http://bit.ly/1a1nFE6>.

наборе DataFrame на наличие строк, содержащих искомое слово, которое мы сохранили в переменной `search_term`. Именованный аргумент `case=False` гарантирует, что мы получим результат в любом случае, независимо от регистра символов, которыми записано слово «raptor».

Переменная `query` — это серия значений `True` и `False`, при передаче которой объекту `df` типа DataFrame мы получим записи, соответствующие значению `True` в нашем поисковом запросе (то есть записи, в которых тема или текст письма содержат слово «raptor»). Эти записи сохраняются в переменной `results`.

Цикл `for`, следующий далее в примере, затем выведет первые 10 совпавших записей. Для вывода первых пяти строк из текста каждого найденного письма здесь использована библиотека `textwrap`. Такое решение позволяет быстро пробежать взглядом вывод. Далее приводится сокращенный ввод результатов:

```
SUBJECT: RE: Pricing of restriction on Enron stock
```

```
-----
```

```
Vince, I just spoke with Rakesh. I believe that there is some
confusion regarding which part of that Raptor transaction we are
talking about. There are actually two different sets of forwards: one
for up to 18MM shares contingently based on price as an offset to the
Whitewing forward shortfall, and the other was for 12MM shares [...]
```

```
SUBJECT: FW: Note on Valuation
```

```
-----
```

```
Vince, I have it. Rakesh -----Original Message----- From: Kaminski,
Vince J Sent: Monday, October 22, 2001 2:39 PM To: Bharati, Rakesh;
Shanbhogue, Vasant Cc: Kaminski, Vince J;
'kimberly.r.scardino@us.andersen.com' Subject: RE: Note on Valuation
Rakesh, I have informed Ryan Siurek (cc Rick Buy) on Oct 4 that [...]
```

```
SUBJECT: FW: Raptors
```

```
-----
```

```
I am forwarding a copy of a message I sent some time ago to the same
address. The lawyer representing the Special Committee (David Cohen)
could not locate it. The message disappeared as well from my mailbox.
Fortunately, I have preserved another copy. Vince Kaminski
-----Original Message----- From: VKaminski@aol.com@ENRON [...]
```

```
SUBJECT: Raptors
```

```
-----
```

```
David, I am forwarding to you, as promised, the text of the
10/04/2001 message to Ryan Siurek regarding Raptor valuations. The
message is stored on my PC at home. It disappeared from my mailbox on
the Enron system. Vince Kaminski *****
***** Subj: FW: [...]
```

Эти отрывки содержат фрагменты информации, складывающиеся в мозаику. Теперь у вас есть необходимые инструменты и знания, чтобы продолжить поиски и выяснить еще больше.

7.4. Анализ собственных почтовых данных

Корпус почтовых данных Enron служит прекрасной иллюстрацией для главы, посвященной анализу почты, но вам наверняка захочется также исследовать содержимое своего почтового ящика. К счастью, многие клиенты поддерживают возможность экспортирования в формат mbox, что упрощает перевод почтовых данных в формат, пригодный для анализа методами, описанными в этой главе.

Например, в Apple Mail можно экспортировать все содержимое почтового ящика в формат mbox, выбрав пункт **Export Mailbox** (Экспорт почтового ящика) в меню **Mailbox** (Почтовый ящик). Также есть возможность экспортировать отдельные письма, выделяя их и выбирая пункт **Save As** (Сохранить как) в меню **File** (Файл). В открывшемся диалоге сохранения выберите в раскрывающемся списке **Format** (Формат) пункт **Raw Message Source** (Исходный текст сообщения), как показано на рис. 7.5. Другие распространенные почтовые клиенты предлагают аналогичные возможности, нужно лишь потратить немного времени на исследование их меню.

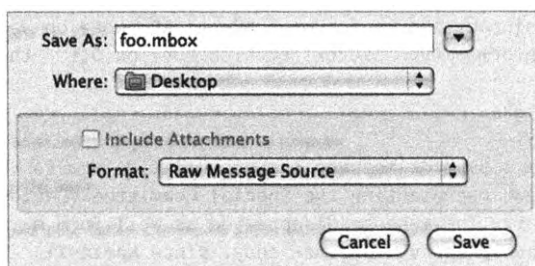


Рис. 7.5. Наиболее распространенные почтовые клиенты предлагают возможность экспортирования почты в формат mbox

Пользующиеся исключительно онлайн-версией почтового клиента могут установить обычный клиент, а затем с его помощью получить всю почту и экспортировать ее, но есть другая возможность, позволяющая полностью автоматизировать процесс создания файла mbox, извлекая данные непосредственно

с сервера. Практически любая служба электронной почты поддерживает протокол POP3¹ (Post Office Protocol, version 3) и многие поддерживают также протокол IMAP², поэтому не составит большого труда написать небольшой сценарий на Python, извлекающий почту с сервера.

Для получения почтовых данных практически откуда угодно можно использовать надежный инструмент командной строки `getmail`³, который написан на Python. Кроме того, в стандартной библиотеке Python имеются два модуля, `poplib`⁴ и `imaplib`⁵, обеспечивающие потрясающую основу для подобных задач, поэтому, поискав в интернете, вы наверняка найдете большое число похожих сценариев. Инструмент `getmail` очень прост в установке и использовании. Например, чтобы получить входящие письма из почтового ящика Gmail, достаточно загрузить и установить этот инструмент и определить настройки в конфигурационном файле `getmailrc`.

В следующем примере демонстрируются настройки для *nix окружения. Пользователям Windows может потребоваться изменить параметры `path` в разделе `[destination]` и `message_log` в разделе `[options]`, определив в них действительные пути, но имейте в виду, что если вам нужно быстрое решение, этот сценарий можно запустить в виртуальной машине с примерами для этой книги:

```
[retriever]
type = SimpleIMAPSSLRetriever
server = imap.gmail.com
username = ptwoobrussell
password = xxx

[destination]
type = Mboxrd
path = /tmp/gmail.mbox

[options]
verbose = 2
message_log = ~/.getmail/gmail.log
```

Определив настройки, просто запустите команду `getmail` в окне терминала, а все остальное она сделает сама. Получив файл `mbox`, вы сможете проанализи-

¹ <http://bit.ly/1a1nHvx>.

² <http://bit.ly/2MXFFvF>.

³ <http://bit.ly/1a1nKaL>.

⁴ <http://bit.ly/1a1nI2G>.

⁵ <http://bit.ly/1a1nIj5>.

зировать его, используя приемы, описанные в этой главе. Вот как выглядит вывод команды `getmail`, извлекающей данные из почтового ящика:

```
$ getmail
getmail version 4.20.0
Copyright (C) 1998-2009 Charles Cazabon. Licensed under the GNU GPL version 2.
SimpleIMAPSSLRetriever:ptwobrussell@imap.gmail.com:993:
msg      1/10972 (4227 bytes) from ... delivered to Mboxrd /tmp/gmail.mbox
msg      2/10972 (3219 bytes) from ... delivered to Mboxrd /tmp/gmail.mbox
...
```

7.4.1. Доступ к почтовому ящику Gmail с использованием OAuth

В начале 2010 года компания Google объявила о начале поддержки доступа к почтовым ящикам Gmail по протоколам IMAP и SMTP с использованием OAuth¹. Это стало знаменательным событием, потому что поддержка OAuth открыла дверь в «Gmail как платформу», позволив сторонним разработчикам создавать приложения, способные извлекать почту из ящиков Gmail и не требующие передавать им имя пользователя и пароль. В этом разделе мы не будем разбирать тонкости работы протокола OAuth 2.0² (краткое введение в OAuth вы найдете в приложении Б) и сосредоточимся на конкретных шагах, перечисленных ниже, которые нужно выполнить, чтобы вы могли получить доступ к данным в своем почтовом ящике Gmail:

1. Откройте в браузере консоль разработчика Google Developer Console³ и создайте или выберите проект. Включите Gmail API.
2. Перейдите на вкладку **Credentials** (Учетные данные), щелкните на раскрывающемся списке **Create credentials** (Создать учетные данные) и выберите пункт **OAuth client ID** (Идентификатор клиента OAuth).
3. В списке **Application type** (Тип приложения) выберите пункт **Other** (Другие типы), введите имя «Gmail API Quickstart» и щелкните на кнопке **Create** (Создать).
4. В открывшемся диалоге щелкните на кнопке **OK**.
5. Щелкните на кнопке загрузки (с изображением стрелки вниз) рядом с вновь созданными учетными данными, чтобы скачать JSON-файл.

¹ <http://bit.ly/1a1nIzH>.

² <http://bit.ly/2GoT1Pl>.

³ <http://bit.ly/2ImRDZM>.

6. Переместите этот файл в свой рабочий каталог и переименуйте его в `client_secret.json`.

Далее нужно установить клиентскую библиотеку Google, что легко сделать с помощью `pip`:

```
pip install --upgrade google-api-python-client
```

Инструкции по установке библиотеки можно найти на ее странице¹. Теперь, когда библиотека установлена и учетные данные сохранены на локальном компьютере, можно начинать писать код, использующий Gmail API. В примере 7.14 приводится код, взятый из руководства Python Quickstart², описывающего порядок использования Gmail API. Сохраните его в файле с именем `quickstart.py`. Если ваш файл `client_secrets.json` находится в том же рабочем каталоге, никаких проблем не должно возникнуть: сценарий откроет в браузере страницу, где вам будет предложено выполнить вход в Gmail и дать разрешение вашему проекту прочитать почту. Код в примере просто выводит ярлыки Gmail (если вы их создали).

Пример 7.14. Подключение к Gmail с использованием OAuth

```
import httplib2
import os

from apiclient import discovery
from oauth2client import client
from oauth2client import tools
from oauth2client.file import Storage

try:
    import argparse
    flags = argparse.ArgumentParser(parents=[tools.argparser]).parse_args()
except ImportError:
    flags = None

# При изменении привилегий удалите прежде сохраненные учетные данные
# в ~/.credentials/gmail-python-quickstart.json
SCOPES = 'https://www.googleapis.com/auth/gmail.readonly'
CLIENT_SECRET_FILE = 'client_secret.json'
APPLICATION_NAME = 'Gmail API Python Quickstart'

def get_credentials():
    """Загружает действительные учетные данные пользователя из хранилища.
```

¹ <http://bit.ly/2EcUP7P>.

² <http://bit.ly/2GNhSvy>.

Если ничего не сохранялось или если учетные данные недействительны, выполнится процедура OAuth2 для получения новых учетных данных.

Возвращает:

Полученные учетные данные.

"""

```
home_dir = os.path.expanduser('~')
credential_dir = os.path.join(home_dir, '.credentials')
if not os.path.exists(credential_dir):
    os.makedirs(credential_dir)
credential_path = os.path.join(credential_dir,
                               'gmail-python-quickstart.json')

store = Storage(credential_path)
credentials = store.get()
if not credentials or credentials.invalid:
    flow = client.flow_from_clientsecrets(CLIENT_SECRET_FILE, SCOPES)
    flow.user_agent = APPLICATION_NAME
    if flags:
        credentials = tools.run_flow(flow, store, flags)
    else: # Needed only for compatibility with Python 2.6
        credentials = tools.run(flow, store)

    print('Storing credentials to ' + credential_path)
return credentials
```

def main():

"""Демонстрирует основы использования Gmail API.

Создает объект для доступа к службе Gmail API и выводит список ярлыков из учетной записи Gmail.

"""

```
credentials = get_credentials()
http = credentials.authorize(httplib2.Http())
service = discovery.build('gmail', 'v1', http=http)

results = service.users().labels().list(userId='me').execute()
labels = results.get('labels', [])

if not labels:
    print('No labels found.')
else:
    print('Labels:')
    for label in labels:
        print(label['name'])

if __name__ == '__main__':
    main()
```

Пример 7.14 можно использовать как заготовку для создания более сложных приложений для доступа к входящей почте в вашем почтовом ящике Gmail. Получив программный доступ к своему почтовому ящику, можно приступить к извлечению и парсингу сообщений. Самое замечательное, что данные можно сохранить в том же формате, с которым вы работали на протяжении всей этой главы, поэтому все ваши сценарии и инструменты будут одинаково хорошо работать и с корпусом Enron, и с вашими собственными почтовыми данными!

7.4.2. Извлечение и парсинг электронных писем

Протокол IMAP — весьма привередливое и сложное создание, но не отчаивайтесь, потому что для поиска и извлечения электронных писем не требуется знать все его тонкости. Кроме того, в интернете можно найти хорошие примеры¹ использования библиотеки `imaplib`.

Следуя за примером реализации доступа к Gmail, можно продолжить использовать OAuth для обращения к ящику с входящей почтой и извлечь из него письма, соответствующие некоторому критерию. Пример 7.15 основан на примере 7.14 и предполагает, что вы уже прошли этап настройки проекта в Google Developer Console и активировали поддержку Gmail API.

Допустим, вы решили отыскать все входящие письма, содержащие слово «Alaska». Код в примере 7.15 вернет первые 10 результатов (изменить это ограничение можно в переменной `max_results`). Затем цикл `for` извлечет из каждого результата идентификатор сообщения и по этому идентификатору загрузит само сообщение.

Пример 7.15. Поиск входящих писем и вывод их содержимого

```
import httpplib2
import os

from apiclient import discovery
from oauth2client import client
from oauth2client import tools
from oauth2client.file import Storage

# При изменении привилегий удалите прежде сохраненные учетные данные
# в ~/.credentials/gmail-python-quickstart.json
```

¹ <http://bit.ly/1a1nJDG>.

```

SCOPES = 'https://www.googleapis.com/auth/gmail.readonly'
CLIENT_SECRET_FILE = 'client_secret.json'
APPLICATION_NAME = 'Gmail API Python Quickstart'

def get_credentials():
    """Загружает действительные учетные данные пользователя из хранилища.

    Если ничего не сохранялось или если учетные данные недействительны,
    выполнится процедура OAuth2 для получения новых учетных данных.

    Возвращает:
        Полученные учетные данные.
    """
    home_dir = os.path.expanduser('~')
    credential_dir = os.path.join(home_dir, '.credentials')
    if not os.path.exists(credential_dir):
        os.makedirs(credential_dir)
    credential_path = os.path.join(credential_dir,
                                   'gmail-python-quickstart.json')

    store = Storage(credential_path)
    credentials = store.get()
    if not credentials or credentials.invalid:
        flow = client.flow_from_clientsecrets(CLIENT_SECRET_FILE, SCOPES)
        flow.user_agent = APPLICATION_NAME
        if flags:
            credentials = tools.run_flow(flow, store, flags)
        else: # Needed only for compatibility with Python 2.6
            credentials = tools.run(flow, store)
        print('Storing credentials to ' + credential_path)
    return credentials

credentials = get_credentials()
http = credentials.authorize(httplib2.Http())
service = discovery.build('gmail', 'v1', http=http)
results = service.users().labels().list(userId='me').execute()
labels = results.get('labels', [])

if not labels:
    print('No labels found.')
else:
    print('Labels:')
    for label in labels:
        print(label['name'])

query = 'Alaska'
max_results = 10

# Найти в Gmail сообщения с заданным словом

```

```
results = service.users().messages().list(userId='me', q=query,
                                          maxResults=max_results).execute()

for result in results['messages']:
    print(result['id'])
    # Получить само сообщение
    msg = service.users().messages().get(userId='me', id=result['id'],
                                          format='minimal').execute()
    print(msg)
```

Чтобы узнать, что еще можно делать с сообщениями в Gmail, прочитайте документацию с описанием API¹. Освоив приемы, описанные в этой главе и в документации, вы сможете создавать сложные инструменты для анализа входящей почты в своем почтовом ящике Gmail. Кроме того, в зависимости от уровня поддержки IMAP или OAuth API, вы сможете написать код на Python, обращающийся к другим веб-службам электронной почты.

Благополучно получив текст из входящих сообщений Gmail, его необходимо подготовить для визуализации или обработки с привлечением методов NLP, описанных в главе 6. Впрочем, чтобы подготовить текст к анализу словосочетаний, не требуется прикладывать много усилий. На самом деле результаты, возвращаемые примером 7.15, можно почти без подготовки передать в пример 5.9, чтобы получить список словосочетаний. Хорошим упражнением по визуализации могло бы стать создание графа, отражающего силу связей между сообщениями, вычисляемую по количеству общих биграмм.

7.4.3. Визуализация закономерностей в электронных письмах с помощью Immersion

Для анализа электронной почты есть целый ряд удобных инструментов. Одним из наиболее перспективных, появившихся в последние годы, является инструмент Immersion², разработанный в MIT Media Lab. Он предлагает взглянуть на входящую почту с точки зрения «общности ваших корреспондентов». Вы подключаетесь к своей учетной записи Gmail, Yahoo! или MS Exchange с помощью платформы, и она генерирует граф, опираясь на поля *To*, *From*, *Cc* и время отправки писем, показывающий, кто с кем связан, и другие аспекты. На рис. 7.6 показан пример скриншота.

¹ <http://bit.ly/2pYRRzy>.

² <http://bit.ly/2q2xUaD>.

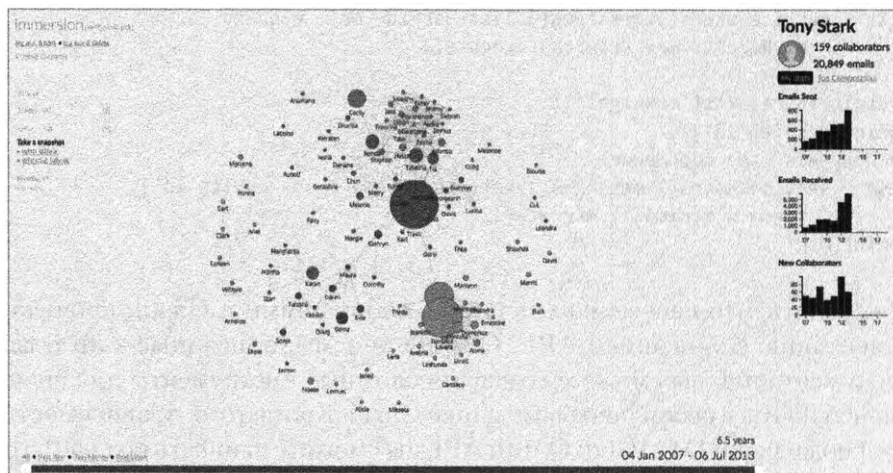


Рис. 7.6. Демонстрация инструмента Immersion визуализации связей, определяемых по электронной почте

Самое примечательное, что вы сможете воспроизвести все виды анализа, предлагаемые этим инструментом, воспользовавшись приемами, изученными в этой и в предыдущих главах, такими как использование библиотеки D3.js на JavaScript или утилит из пакета matplotlib в Jupyter Notebook. В вашем арсенале достаточно сценариев и приемов, которые легко можно применить для анализа и создания похожих визуализаций на основе любых данных, будь то электронные письма, архивы веб-страниц или коллекции твитов. Конечно, вы должны тщательно продумать дизайн приложения, чтобы обеспечить пользователю максимальное удобство, но все строительные блоки, необходимые для создания такого приложения, у вас уже есть.

7.5. Заключительные замечания

В этой главе мы проделали большую работу и получили важные результаты, используя инструменты, представленные в предыдущих главах. Каждая глава последовательно опиралась на предыдущие, рассказывая историю о данных и их анализе, и мы почти достигли конца книги. В этой главе мы лишь слегка затронули возможности обработки почтовых данных, но вы наверняка сможете воспользоваться знаниями, полученными в предыдущих главах, чтобы открыть для себя удивительные подробности о своих социальных связях и личной жизни, проанализировав свою электронную почту, которые добавляют новое захватывающее измерение в анализ.

Основное свое внимание мы сосредоточили на mbox, простом и удобном формате, обеспечивающем высокую переносимость и простоту анализа с применением многих инструментов и пакетов Python, и, надеюсь, вы осознали ценность использования стандартного и переносимого формата для обработки таких сложных источников информации, как почтовые ящики. Существует невероятное количество технологий с открытым исходным кодом, доступных для анализа данных в формате mbox, и Python — потрясающий язык для их создания. Приложив совсем немного усилий, вы сможете сосредоточиться на проблеме.



Исходный код примеров для этой и всех других глав доступен на GitHub¹ в удобном формате Jupyter Notebook, который вы можете опробовать, не покидая веб-браузера.

7.6. Упражнения

- Выберите подмножество почтовых отправок из корпуса Enron для анализа. Например, почитайте о деле Enron в интернете или посмотрите документальный фильм, а затем выберите 10–15 почтовых ящиков, представляющих наибольший интерес, и попробуйте выявить закономерности взаимодействий, используя приемы, представленные в этой главе.
- Примените методы анализа текста, описанные в предыдущих главах, к содержимому электронных писем. Попробуйте выявить основные темы общения между людьми. Какие, на ваш взгляд, достоинства и недостатки имеет полнотекстовый поиск в сравнении с идеями информационного поиска из предыдущих глав?
- Рассмотрите алгоритм определения цепочек электронных писем², который можно использовать как эффективную эвристику для воссоздания диалогов по электронным письмам. Пример реализации³ этого алгоритма доступен в примерах к первому изданию этой книги.
- Используйте проект SIMILE Timeline⁴ для визуализации цепочек сообщений, полученных с помощью вышеупомянутого алгоритма. На сайте проекта Timeline вы найдете минимальный пример⁵ того, что должны реали-

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² <http://bit.ly/1a1nQ23>.

³ <http://bit.ly/1a1nQ2e>.

⁴ <http://bit.ly/1a1nQz3>.

⁵ <http://bit.ly/2Nnj8W1>.

зовать; за информацией о дополнительных возможностях обращайтесь к документации.

- Выполните поиск по слову «Enron» в Google Scholar¹ и прочитайте некоторые статьи и исследования, посвященные этой теме. Используйте их как основу для своих собственных исследований.
- Опробуйте примеры² для этой главы из предыдущего издания книги. В них для индексирования и поиска в корпусе Enron вместо pandas используется MongoDB³. Для взаимодействия с базами данных MongoDB можно использовать удобную библиотеку pymongo⁴. Оцените достоинства и недостатки использования полноценной базы данных вместо библиотеки pandas и структуры DataFrame.
- Прочитайте статью⁵ в блоге астрофизика и исследователя данных Джастина Эллиса (Justin Ellis), который решил проанализировать свой почтовый ящик Gmail с помощью pandas. Если у вас есть почтовый ящик Gmail с достаточно большим количеством писем, попробуйте воспроизвести некоторые визуализации, описанные в статье.
- Напишите код на Python для доступа к вашей электронной почте программным способом. Попробуйте выяснить статистическую информацию о себе самом: когда вы получаете и посылаете больше всего писем? В какие дни недели вы ведете наиболее интенсивную переписку? Или попытайтесь решить чуть более сложную задачу и определить типы отправленных писем с самой высокой вероятностью получения ответа на них. Для этого можно попробовать сравнивать длину писем или время суток, когда они отправлялись.

7.7. Онлайн-ресурсы

Ниже приводится список ссылок, упоминавшихся в этой главе, которые могут оказаться полезными для вас:

- Загрузите свои данные из Google⁶.
- Корпус электронной почты Enron для загрузки⁷.

¹ <http://bit.ly/1a1nR6c>.

² <http://bit.ly/2J9gVMv>.

³ <https://www.mongodb.com/>.

⁴ <http://bit.ly/2uBdISn>.

⁵ <http://bit.ly/2Iimsiq>.

⁶ <https://takeout.google.com/>.

⁷ <http://bit.ly/1a1nmsU>.

- Корпус Enron¹.
- Общедоступные корпус и база данных электронной почты Enron².
- Описание дела Enron³.
- Статьи о Enron в Google Scholar⁴.
- getmail⁵.
- Git для Windows⁶.
- Immersion: взгляд на почту с точки зрения общности ваших корреспондентов⁷.
- Алгоритм Джейми Завински (Jamie Zawinski) для определения цепочек электронных писем⁸.
- Типы MIME⁹.
- Демонстрационные онлайн-примеры SIMILE Timeline¹⁰.
- Статья «Personal Analytics Part 1: Gmail»¹¹.
- Проект SIMILE Timeline¹².
- Статья «Using OAuth 2.0 to Access Google APIs»¹³.

¹ <http://bit.ly/1a1nj01>.

² <http://www.enron-mail.com/>.

³ <http://bit.ly/1a1nuZo>.

⁴ <http://bit.ly/1a1nR6c>.

⁵ <http://bit.ly/1a1nKaL>.

⁶ <http://bit.ly/2Hiaox1>.

⁷ <http://bit.ly/2q2xUaD>.

⁸ <http://bit.ly/1a1nQ23>.

⁹ <http://bit.ly/2Qzxftu>.

¹⁰ <http://bit.ly/1a1nOr1>.

¹¹ <http://bit.ly/2Iimsiq>.

¹² <http://bit.ly/1a1nQz3>.

¹³ <http://bit.ly/2GoT1Pl>.

8

Анализ GitHub: особенности сотрудничества при разработке ПО, графы интересов и многое другое

Быстрое развитие GitHub в последние годы привело к тому, что этот сайт де-факто превратился в социальную платформу для программистов с обманчиво простой предпосылкой: дать разработчикам первоклассное решение для создания и сопровождения программных проектов с открытым исходным кодом, включающее поддержку распределенной открытой системы управления версиями¹ под названием *Git*². В отличие от других систем управления версиями, таких как CVS³ или Subversion⁴, в Git отсутствует каноническая ветвь кода. Все ветви являются рабочими, и разработчики могут вносить локальные изменения в свою рабочую ветвь без необходимости подключаться к центральному серверу.

Парадигма распределенного управления версиями исключительно хорошо подходит для идеи *совместного программирования*, лежащей в основе GitHub, потому что позволяет разработчикам, желающим принять участие в проекте, создать свою рабочую *ветвь* кода и немедленно приступить к работе над ним, точно так же, как разработчик, давший начало проекту. Система Git не только поддерживает семантику свободного ветвления репозитория, но также позволяет относительно легко передавать изменения из *дочернего репозитория* обратно в *родительский*. В пользовательском интерфейсе GitHub этот процесс называется *запросом на включение* (pull request).

Простота этого решения обманчива, и тем не менее возможность создавать и совместно работать над программными проектами с использованием простых и удобных процедур, требующих минимальных накладных расходов (как

¹ <http://bit.ly/1a1o1u8>.

² <http://bit.ly/16mhOep>.

³ <http://bit.ly/1a1nZCI>.

⁴ <http://bit.ly/2GZy78S>.

только вы поймете некоторые основные особенности работы Git), избавила разработчиков от множества утомительных деталей, которые препятствовали инновациям в разработке ПО с открытым исходным кодом, включая удобства, выходящие за рамки визуализации данных и взаимодействий с другими системами. Проще говоря, GitHub можно рассматривать как средство разработки открытого ПО. То есть несмотря на то, что идея совместной разработки не нова, централизованная платформа, такая как GitHub, расширяет возможности сотрудничества и дает беспрецедентную возможность внедрения инноваций, упрощая создание проектов, организуя обмен исходным кодом, поддерживая обратную связь и управление проблемами, принимая исправления и улучшения и многое другое. В последнее время кажется даже, что GitHub все больше ориентируется¹ на тех, кто не является разработчиком, и превращается в одну из самых активных социальных платформ для совместной работы.

Для большей ясности сразу заметим, что эта глава не является руководством по использованию Git или GitHub как распределенной системы управления версиями и вообще не обсуждает программную архитектуру Git. (За этой информацией обращайтесь к онлайн-инструкциям о Git, таким как [gitscm.com](http://git-scm.com)², коих можно найти великое множество в интернете.) В этой главе делается попытка научить вас приемам использования GitHub API для выявления особенностей сотрудничества в области разработки ПО.



Загрузите последний исправленный исходный код примеров для этой (и любой другой) главы, доступный по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Воспользуйтесь также преимуществами виртуальной машины, описанными в приложении А, чтобы получить максимальное удовольствие от опробования примеров кода.

8.1. Обзор

Эта глава знакомит с GitHub, как платформой совместного программирования, и графовыми методами анализа с помощью NetworkX. Здесь вы узнаете, как использовать богатые данные из GitHub, создав графическую модель данных, которая имеет множество разных применений. В частности, мы рассмотрим отношения между пользователями GitHub, репозиториями и языками программирования, представив их в виде графа интересов, который позволяет рассматривать узлы и связи в графе с точки зрения людей и их интересов. В наши

¹ <http://bit.ly/1a1o2OZ>.

² <http://bit.ly/1a1o2hZ>.

дни среди хакеров, предпринимателей и веб-разработчиков широко обсуждается вопрос о том, будет ли в будущем Всемирная паутина основываться на некотором понятии графа интересов, поэтому сейчас самое время познакомиться с этой новой идеей и всем, что она влечет за собой.

В общем и целом, эта глава следует тому же предсказуемому шаблону, что и предыдущие главы, и охватывает следующие темы:

- платформа разработки GitHub и как выполнять запросы к ее API;
- граф-схемы и как моделировать графы свойств с помощью NetworkX;
- граф интересов и как построить такой граф на основе данных из GitHub;
- применение NetworkX для выполнения запросов к графу свойств;
- алгоритмы вычисления центральности графа, включая центральность по степени (degree), по посредничеству (betweenness) и по близости (closeness).

8.2. GitHub API

Подобно другим социальным сетям, описанным в этой книге, GitHub предлагает разработчикам обширную документацию с описанием своих программных интерфейсов (API) и условиями, регулирующими их использование, с примерами кода и множеством других сведений. Несмотря на богатство API, мы сосредоточимся лишь на нескольких вызовах, необходимых для получения данных, на основе которых мы будем создавать графы интересов, связывающие разработчиков, проекты, языки программирования и другие аспекты, имеющие отношение к созданию программного обеспечения. Программные интерфейсы дают почти все, что нужно, чтобы вызвать приятные впечатления у пользователей, которые предлагает сам сайт github.com¹: с их помощью вы сможете создавать привлекательные и, может быть, даже прибыльные приложения.

Основными единицами для GitHub являются *пользователи* и *проекты*. Если вы читаете эту страницу, вам, возможно, уже довелось извлечь исходный код примеров для этой книги со страницы проекта² на сайте GitHub, поэтому далее будет предполагаться, что вы уже посетили несколько страниц проектов на GitHub, покопались в них и в целом представляете, что предлагает GitHub.

¹ <http://bit.ly/1a1kFHM>.

² <http://bit.ly/Mining-the-Social-Web-3E>.

Пользователь GitHub имеет общедоступный профиль, обычно включающий один или несколько репозиториев с кодом, либо созданных им, либо ответвленных от репозиториев других пользователей GitHub. Например, пользователь `ptwobrussell`¹ владеет парой репозиториев на GitHub, один из которых называется `Mining-the-Social-web`², а другой `Mining-the-Social-web-2nd-Edition`³. Кроме того, пользователь `ptwobrussell` создал несколько ответвлений репозиториев других пользователей, чтобы получить рабочие срезы проектов для целей разработки, и эти ответвленные проекты также можно видеть в его профиле.

Привлекательность GitHub отчасти объясняется тем, что `ptwobrussell`, как и любой другой пользователь, может делать с этими ответвлениями проектов все, что угодно (в рамках их лицензионных соглашений). Когда пользователь отвечает репозиторий, он фактически вступает во владение рабочей копии того репозитория и может делать с ней все, что угодно: просто копаться в коде, существенно изменить его и создать отдельную ветвь оригинального проекта, которая никогда не сольется обратно с родительским репозиторием. Большинство ответвлений никогда не превращаются в самостоятельные проекты, и создать такое ответвление не представляет никакого труда с точки зрения управления исходным кодом. Ответвление может быть короткоживущим и быстро сливаться с родительской ветвью или долгоживущим и превратиться в совершенно отдельный проект с собственным сообществом. Барьер вступления в разработку программного обеспечения с открытым исходным кодом и в другие проекты, которые все чаще появляются на GitHub, действительно очень низок.

Помимо создания проектов пользователь GitHub может также добавлять в закладки другие проекты, отмечая их звездами, становясь так называемым *звездочетом* проекта. Создать закладку на проект — все равно что создать закладку на веб-страницу или твит. Вы обозначаете свой интерес к проекту, и он появляется в вашем списке закладок на сайте GitHub для быстрого доступа. Обычно сразу же бросается в глаза, что ответвления проектов создаются намного реже, чем закладки. Закладки — широко известное понятие, закрепившееся в нашем сознании за десятилетия веб-серфинга, тогда как понятие ветвления кода подразумевает намерение изменить его или внести свой вклад в его разработку. В оставшейся части этой главы мы сосредоточимся в основном на использовании списка звездочетов проекта в качестве основы для создания графа интересов.

¹ <http://bit.ly/1a1o4GC>.

² <http://bit.ly/1a1o6Ow>.

³ <http://bit.ly/1a1kNqy>.

8.2.1. Подключение к GitHub API

Так же как другие социальные сети, GitHub поддерживает протокол OAuth, и для доступа к API вы должны создать аккаунт и либо создать приложение для использования в качестве потребителя услуг API, либо создать «личный» ключ, который будет привязан непосредственно к вашей учетной записи. В этой главе мы будем использовать личный ключ доступа, который легко получить, щелкнув на кнопке в разделе Personal Access API Tokens (Личные ключи доступа к API) в меню Applications¹ (Приложения) вашей учетной записи, как показано на рис. 8.1. (Более подробный обзор протокола OAuth вы найдете в приложении Б.)

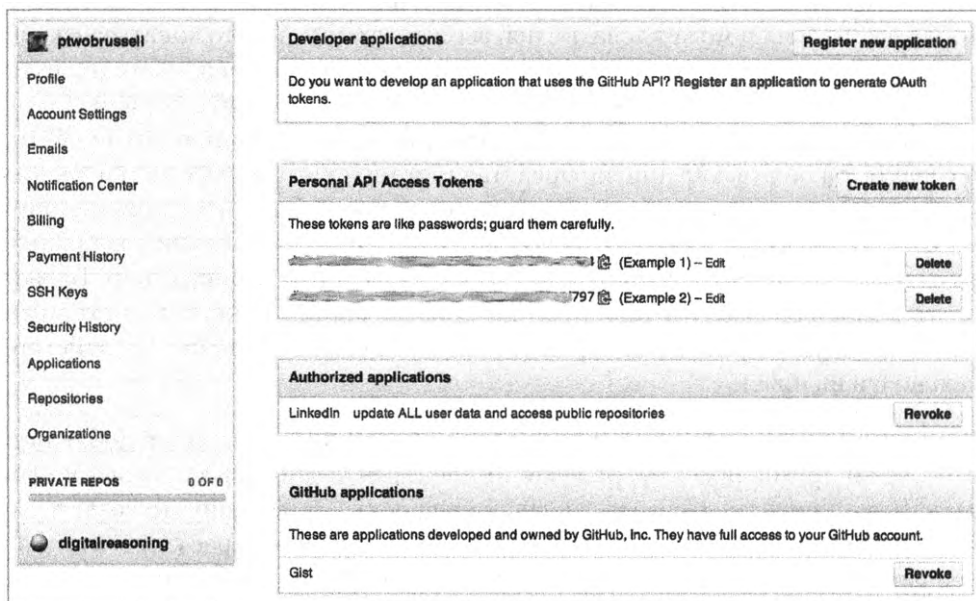


Рис. 8.1. Создание личного ключа доступа к API в меню Applications (Приложения) в учетной записи и добавление осмысленного примечания, описывающего цель получения ключа

Ключ доступа можно получить не только в пользовательском интерфейсе GitHub, но и программно, как иллюстрирует код в примере 8.1, являющийся адаптацией примера получения ключа из командной строки, описанного

¹ <http://bit.ly/1a1o7lw>.

в статье «Creating a personal access token for the command line»¹ на сайте GitHub Help. (Если вы решили не пользоваться виртуальной машиной для этой книги, которая описывается в приложении А, тогда перед запуском этого примера выполните команду `pip install requests`.)

Пример 8.1. Программный способ получения личного ключа доступа к GitHub API

```
import requests
import json

username = '' # укажите здесь ваше имя пользователя GitHub
password = '' # укажите здесь свой пароль для GitHub

# Обратите внимание, что учетные данные передаются через
# защищенное SSL-соединение
url = 'https://api.github.com/authorizations'
note = 'Mining the Social Web - Mining Github'
post_data = {'scopes': ['repo'], 'note': note }

response = requests.post(
    url,
    auth = (username, password),
    data = json.dumps(post_data),
)

print("API response:", response.text)
print()
print("Your OAuth token is", response.json()['token'])

# Отозвать этот ключ можно по адресу https://github.com/settings/tokens
```

Так же как во многих других социальных сетях, программный интерфейс GitHub действует поверх протокола HTTP и доступен из программ на любом языке программирования, где можно выполнять HTTP-запросы, включая инструменты командной строки. Но следуя практике, сложившейся в предыдущих главах, мы решили использовать библиотеку для Python, чтобы избежать рутины, связанной с отправкой запросов, парсингом ответов и постраничным приемом результатов. В данном конкретном случае мы будем использовать библиотеку PyGithub², которую можно установить с помощью команды `pip install PyGithub`. Для начала рассмотрим пару примеров, демонстрирующих выполнение запросов к GitHub API, а затем перейдем к обсуждению графических моделей.

¹ <http://bit.ly/1a1o7IG>.

² <http://bit.ly/1a1o7Ca>.

В этой главе мы создадим граф интересов для GitHub-репозитория Mining-the-Social-Web¹ и определим связи между ним и его звездочетами. Список звездочетов можно получить с помощью List Stargazers API². Чтобы получить представление о том, как выглядит ответ, возвращаемый на запрос к API, можете попробовать скопировать и вставить в адресную строку веб-браузера следующий URL: <https://api.github.com/repos/ptwobrussell/Mining-the-Social-Web/stargazers>.



Несмотря на то что вы читаете третье издание книги, в этой главе мы продолжим использовать репозиторий с кодом примеров для первого издания, который к моменту написания этих строк уже был отмечен звездами более 1000 раз. Анализ любого репозитория, включая репозитории для второго и третьего изданий этой книги, легко выполнить, просто изменив имя проекта в примере 8.3.

Возможность посылать неаутентифицированные запросы подобным способом очень удобна для исследования API, а ограничение в 60 неаутентифицированных запросов в час не кажется слишком строгим и позволяет экспериментировать и исследовать результаты. Вы можете добавить в URL строку запроса в форме `?access_token=xxx`, где `xxx` — ключ доступа, и выполнить аутентифицированный запрос. GitHub позволяет выполнять до 5000 аутентифицированных запросов в час, как утверждается в документации для разработчиков в разделе с описанием ограничений³. Пример 8.2 демонстрирует выполнение запроса и получение ответа. (Имейте в виду, что в ответ на запрос возвращается только первая страница с результатами, при этом в заголовках HTTP поставляется информация, необходимая для навигации по страницам, как описывается в разделе документации для разработчиков, посвященном постраничному извлечению результатов⁴.)

Пример 8.2. Выполнение прямого HTTP-запроса к GitHub API

```
import json
import requests

# Неаутентифицированный запрос не содержит строки запроса ?access_token=xxx
url = "https://api.github.com/repos/ptwobrussell/Mining-the-Social-Web/stargazers"
response = requests.get(url)
```

¹ <http://bit.ly/1a1o6Ow>.

² <http://bit.ly/1a1o9dd>.

³ <http://bit.ly/1a1oblo>.

⁴ <http://bit.ly/1a1o9Ki>.

```
# Вывести информацию о первом звездочете
print(json.dumps(response.json()[0], indent=1))
print()
```

```
# Вывести заголовки
for (k,v) in response.headers.items():
    print(k, "=>", v)
```

Далее приводится пример вывода этого сценария:

```
{
  "login": "rdempsey",
  "id": 224,
  "avatar_url": "https://avatars2.githubusercontent.com/u/224?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/rdempsey",
  "html_url": "https://github.com/rdempsey",
  "followers_url": "https://api.github.com/users/rdempsey/followers",
  "following_url": "https://api.github.com/users/rdempsey/following{/other_user}",
  "gists_url": "https://api.github.com/users/rdempsey/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/rdempsey/starred{/owner}/{repo}",
  "subscriptions_url": "https://api.github.com/users/rdempsey/subscriptions",
  "organizations_url": "https://api.github.com/users/rdempsey/orgs",
  "repos_url": "https://api.github.com/users/rdempsey/repos",
  "events_url": "https://api.github.com/users/rdempsey/events{/privacy}",
  "received_events_url": "https://api.github.com/users/rdempsey/received_events",
  "type": "User",
  "site_admin": false
}
```

```
Server => GitHub.com
Date => Fri, 06 Apr 2018 18:41:57 GMT
Content-Type => application/json; charset=utf-8
Transfer-Encoding => chunked
Status => 200 OK
X-RateLimit-Limit => 60
X-RateLimit-Remaining => 55
X-RateLimit-Reset => 1523042441
Cache-Control => public, max-age=60, s-maxage=60
Vary => Accept
ETag => W/"b43b2c639758a6849c9f3f5873209038"
X-GitHub-Media-Type => github.v3; format=json
Link => <https://api.github.com/repositories/1040700/stargazers?page=2>;
rel="next", <https://api.github.com/repositories/1040700/stargazers?page=39>;
rel="last"
Access-Control-Expose-Headers => ETag, Link, Retry-After, X-GitHub-OTP,
X-RateLimit-Limit, X-RateLimit-Remaining, X-RateLimit-Reset,
X-OAuth-Scopes, X-Accepted-OAuth-Scopes, X-Poll-Interval
Access-Control-Allow-Origin => *
```

```
Strict-Transport-Security => max-age=31536000; includeSubdomains; preload
X-Frame-Options => deny
X-Content-Type-Options => nosniff
X-XSS-Protection => 1; mode=block
Referrer-Policy => origin-when-cross-origin, strict-origin-when-cross-origin
Content-Security-Policy => default-src 'none'
X-Runtime-rack => 0.057438
Content-Encoding => gzip
X-GitHub-Request-Id => ADE2:10F6:8EC26:1417ED:5AC7BF75
```

Как видите, GitHub возвращает массу полезной информации не в теле HTTP-ответа, а в HTTP-заголовках. Подробное описание всех этих заголовков вы найдете в документации для разработчиков, тем не менее отметим заголовок `status`, код 200 в котором сообщает, что запрос обработан успешно; заголовки, имеющие отношение к ограничениям, такие как `x-ratelimit-remaining`; и заголовок `link`, содержащий строки, такие как

```
https://api.github.com/repositories/1040700/stargazers?page=2; rel="next",
https://api.github.com/repositories/1040700/stargazers?page=29; rel="last"
```

с предварительно подготовленными URL, первый из которых можно использовать для получения следующей страницы с результатами, а второй — как индикатор, определяющий общее число страниц.

8.2.2. Выполнение запросов к GitHub API

Для выполнения запросов можно, конечно, использовать такую библиотеку, как `requests`, и вручную извлекать из ответов всю необходимую информацию. Но мы пойдем другим путем и используем библиотеку `PyGithub`, которая значительно упрощает задачу и скрывает детали реализации GitHub API, давая возможность оставаться в рамках чистого Python API. И что самое замечательное, если GitHub изменит реализацию своего API, наш код, использующий `PyGithub`, останется работоспособным.

Прежде чем начать посылать запросы с помощью `PyGithub`, уделим немного времени исследованию тела самого ответа. Оно содержит довольно много информации, но наибольший интерес для нас представляет поле `login`, содержащее имя пользователя GitHub, который поставил звезду рассматриваемому репозиторию. Эта информация может служить основой для запросов ко многим другим GitHub API, таким как «List repositories being starred»¹ (список репо-

¹ <http://bit.ly/1a1oc8X>.

зиториев, получивших звезды от пользователя), возвращающий список всех репозиториев, которые оценил данный пользователь. Этот API открывает перед нами широкие возможности — начав с произвольного репозитория и запросив список заинтересовавшихся им пользователей, мы можем затем запросить дополнительные репозитории, оцененные этими пользователями, и выявить любые имеющиеся закономерности.

Например, разве не интересно узнать, какой репозиторий будет следующим по числу сделанных на него закладок среди пользователей, добавивших закладку на Mining-the-Social-Web? Ответ на этот вопрос мог бы служить обоснованной рекомендацией для пользователей GitHub, и не нужно обладать богатым воображением, чтобы представить области, где такие рекомендации могли бы улучшить и поспособствовать улучшению впечатления пользователей от приложений, как, например, в Amazon и Netflix. Графы интересов по своей природе способны вырабатывать такие рекомендации, и это одна из причин, почему в последнее время графы интересов стали такой жаркой темой для обсуждения в определенных кругах.

Пример 8.3 демонстрирует, как с помощью PyGithub получить список всех звездочетов репозитория, чтобы на его основе можно было начать строить граф интересов.

Пример 8.3. Получение списка звездочетов репозитория с помощью PyGithub

```
from github import Github # pip install pygithub

# Укажите здесь свой ключ доступа

ACCESS_TOKEN = ''

# Укажите имя пользователя и его репозиторий
USER = 'ptwobrussell'
REPO = 'Mining-the-Social-Web'
#REPO = 'Mining-the-Social-Web-2nd-Edition'

client = Github(ACCESS_TOKEN, per_page=100)
user = client.get_user(USER)
repo = user.get_repo(REPO)

# Получить список пользователей, добавивших репозиторий в закладки.
# Поскольку обратно возвращается итератор, выполняющий отложенные операции,
# необходимо явно выполнить его обход, чтобы получить общее число звездочетов.

stargazers = [ s for s in repo.get_stargazers() ]
print("Number of stargazers", len(stargazers))
```

Библиотека `PyGithub` сама заботится о деталях реализации API, предоставляя несколько удобных объектов для выполнения запросов. В данном случае мы создаем соединение с GitHub и в именованном параметре `per_page` сообщаем желаемое максимальное число результатов на одной странице (100), отличающееся от числа по умолчанию (30). Затем получаем репозиторий конкретного пользователя и запрашиваем число звезд, присвоенных этому репозиторию. Пользователи могут владеть репозиториями с одинаковыми названиями, поэтому нет возможности запросить конкретный репозиторий только по его названию. Кроме того, имена пользователей и названия репозиториев могут совпадать, поэтому следует проявить особую осторожность и указать тип запрашиваемого объекта, обращаясь к GitHub API и передавая имена в качестве идентификаторов. Мы учтем все это при создании графов с узлами, имена которых могут интерпретироваться неоднозначно, если явно не указать, что они являются именами пользователей или названиями репозиториев.

Наконец, обычно `PyGithub` возвращает ленивые итераторы. В данном случае это означает, что запрос не вернет сразу все 29 страниц с результатами. Вместо этого итератор будет ждать запроса конкретной страницы в очередной итерации и только потом извлечет ее. По этой причине нам пришлось явно исчерпать ленивый итератор в генераторе списка, чтобы получить действительное число звездочетов.

В целом программный интерфейс `PyGithub` довольно близко имитирует GitHub API, подробное описание которого вы найдете в документации¹ к библиотеке. Также имеется встроенная документация `pydoc`, доступная в интерактивной оболочке Python с помощью таких функций, как `dir` и `help`. Кроме этого, поддержка автоматического дополнения и «волшебный знак вопроса» в IPython или Jupyter Notebook помогут вам выяснить, какие методы доступны для того или иного объекта. Прежде чем продолжить, найдите время и поэкспериментируйте с GitHub API и `PyGithub`, чтобы лучше познакомиться с имеющимися возможностями. Например, для проверки своих навыков попробуйте обойти всех звездочетов `Mining-the-Social-Web` (или некоторое их подмножество) и выполните простейший частотный анализ, определяющий, какие другие репозитории могут заинтересовать вас. Выполнить этот анализ вам помогут `collections.Counter` или `nltk.FreqDist`.

¹ <http://bit.ly/2qaoCtT>.

8.3. Моделирование данных с помощью графов свойств

В главе 2 графы были описаны как средство представления, анализа и визуализации социальных данных из Facebook. Этот раздел содержит более полное их рассмотрение и, как мы надеемся, послужит полезным введением в графовые вычисления. Практика использования графов продолжает быстро развиваться, потому что графы являются естественной абстракцией для моделирования многих явлений реального мира. Графы обладают большой гибкостью представления данных в экспериментах и анализе, которую трудно превзойти с использованием других средств, таких как реляционные базы данных. Анализ графов, безусловно, не является панацеей, но владение приемами моделирования данных в виде графов станет важным дополнением к инструментам в вашем арсенале.



Введение в теорию графов выходит далеко за рамки этой главы, поэтому в последующем обсуждении мы просто попытаемся познакомить вас с некоторыми ключевыми ее понятиями. Если вы захотите получить чуть более глубокие познания, прежде чем продолжить, посмотрите видеолекцию¹ «Graph Theory — An Introduction!» на YouTube.

В оставшейся части раздела будет представлена распространенная разновидность графов с названием *граф свойств*, которую мы используем для моделирования данных из GitHub в виде графа интересов с помощью пакета NetworkX² для Python. Граф свойств — это структура данных, в которой сущности представлены *узлами*, а отношения между сущностями — *ребрами*. Каждая вершина (узел) имеет уникальный идентификатор, набор свойств в виде пар ключ/значение и коллекцию ребер. Ребра, в свою очередь, уникально определяются парой соединяемых ими узлов и тоже могут иметь свойства.

На рис. 8.2 изображен тривиальный граф свойств с двумя узлами, которые уникально идентифицируются как X и Y и связаны неописанным отношением. Данный конкретный граф называется *ориентированным графом*, потому что его ребра имеют направление (ориентированы), что встречается не всегда, если только ребро не имеет корня, определяющего его ориентированность в смысле моделируемой предметной области.

¹ <http://bit.ly/1a1odto>.

² <http://bit.ly/1a1ocFV>.



Рис. 8.2. Тривиальный граф свойств с ориентированными ребрами

Тривиальный граф свойств можно выразить в коде с помощью NetworkX, как показано в примере 8.4. (Установить этот пакет можно командой **pip install networkx**, если вы не используете готовую виртуальную машину с примерами для этой книги.)

Пример 8.4. Создание тривиального графа свойств

```
import networkx as nx # pip install networkx

# Создать ориентированный граф
g = nx.DiGraph()

# Добавить ориентированное ребро, соединяющее X и Y
g.add_edge('X', 'Y')

# Вывести некоторую информацию о графе
print(nx.info(g))

# Получить узлы и ребра из графа
print("Nodes:", g.nodes())
print("Edges:", g.edges())
print()

# Получить свойства узлов
print("X props:", g.node['X'])
print("Y props:", g.node['Y'])
print()

# Получить свойства ребра
print("X=>Y props:", g['X']['Y'])
print()

# Изменить свойства узла
g.node['X'].update({'prop1' : 'value1'})
print("X props:", g.node['X'])
```

```
print()

# Изменить свойства ребра

g['X']['Y'].update({'label' : 'label1'})
print("X=>Y props:", g['X']['Y'])
```

Этот пример выведет следующее:

```
Name:
Type: DiGraph
Number of nodes: 2
Number of edges: 1
Average in degree: 0.5000
Average out degree: 0.5000

Nodes: ['Y', 'X']
Edges: [('X', 'Y')]

X props: {}
Y props: {}
X=>Y props: {}

X props: {'prop1': 'value1'}

X=>Y props: {'label': 'label1'}
```

В данном конкретном примере метод `add_edge` ориентированного графа добавляет ребро, связывающее уникально идентифицируемые узлы `X` и `Y` и направленное от узла `X` к узлу `Y`, в результате чего получается граф с двумя узлами и ребром между ними. Это ребро уникально идентифицируется кортежем `(X, Y)`, потому что оба узла, которые оно связывает, уникально идентифицируют себя. Обратите внимание, что операция добавления ребра с направлением из `Y` в `X` создаст второе ребро в графе, и это второе ребро будет обладать своим набором свойств. Обычно нет необходимости создавать это второе ребро, потому что есть возможность получить входящие и исходящие ребра узла и выполнять переходы по ребрам в любом направлении, но иногда бывает удобно явно включить в граф такие дополнительные ребра.

Степенью (degree) узла графа называют количество ребер, связывающих его с другими узлами, а для ориентированных графов существуют понятия *степень входа* (in degree) и *степень выхода* (out degree), потому что ребра в этих графах имеют направление. Среднее степени входа и среднее степени выхода дают нормализованные оценки, которые представляют количество узлов, имеющих входящие и исходящие ребра. В данном конкретном случае ориентированный

граф имеет одно направленное ребро, поэтому в нем имеется один узел с исходящим и один узел с входящим ребром.

Степени входа и выхода узла — это фундаментальные понятия в теории графов. Если предположить, что число вершин в графе известно, тогда среднее степени дает меру *плотности* (density) графа: отношение числа фактически имеющихся ребер к числу потенциально возможных ребер, если бы граф был полносвязанным, или полным. В полном графе каждый узел связан с каждым другим узлом, а для случая ориентированного графа это означает, что все узлы имеют входящие ребра от всех других узлов.

ПОЯВЛЕНИЕ БОЛЬШИХ ГРАФОВЫХ БАЗ ДАННЫХ

Эта глава знакомит с графами свойств, универсальными структурами данных, которые можно использовать для моделирования сложных сетей с использованием простых примитивов — узлов и ребер. Моделирование будет осуществляться в соответствии с гибкой схемой, основанной на интуитивном понимании, и для подобных узких областей такого прагматического подхода часто оказывается вполне достаточно. Как будет показано далее в этой главе, графы свойств обеспечивают удивительную гибкость и универсальность в моделировании и обработке сложных данных.

NetworkX, библиотека на языке Python для работы с графами, используемая в этой книге, включает набор мощных инструментов для моделирования графов свойств. Но имейте в виду, что NetworkX реализует графовую базу данных, размещаемую в памяти. Ваши возможности в этом случае прямо пропорциональны объему оперативной памяти в вашем компьютере. Во многих ситуациях ограничения, обусловленные нехваткой памяти, можно обойти, используя меньший набор данных или компьютер с большим объемом памяти. Однако для все чаще возникающих ситуаций, когда в обработку вовлечены так называемые «большие данные» и используются базы данных Hadoop и NoSQL, графы в оперативной памяти просто не годятся.

Среднее степени входа для всего графа вычисляется как сумма степеней входа всех узлов, деленная на общее число узлов в графе, то есть для случая в примере 8.4 это будет 1, деленное на 2. Среднее степени выхода вычисляется точно так же, с той лишь разницей, что на число узлов в графе делится сумма степеней выхода. При работе с ориентированными графами вы всегда будете получать равное число входящих и исходящих ребер, потому что каждое ребро связывает только два узла¹, и по этой же причине средние степеней входа и выхода тоже будут равны.

¹ Более абстрактная версия графа, называемая *гиперграфом* (<http://bit.ly/1a1ocWm>), содержит *гиперребра*, которые могут связывать произвольное число вершин.



В общем случае максимальное значение среднего степени входа и выхода на единицу меньше числа узлов в графе. Найдите минуту, чтобы убедиться в этом, определив необходимое число ребер для получения полного графа.

В следующем разделе мы сконструируем граф интересов, используя те же примитивы графов свойств, и покажем дополнительные методы для работы с реальными данными. Но сначала поэкспериментируйте с графом, попробовав добавить в него несколько узлов, ребер и свойств. В документации¹ к NetworkX приводится множество вводных примеров, которые вы также можете изучить, если это ваше первое знакомство с графами и вам нужны дополнительные наглядные примеры.

8.4. Анализ графов интересов в GitHub

Теперь, вооружившись инструментами для выполнения запросов к GitHub API и моделирования возвращаемых данных в виде графов, применим наши знания и начнем конструировать и анализировать граф интересов. Начнем с репозитория, представляющего интерес для группы пользователей GitHub, и с помощью GitHub API выявим всех, кто дал звезду этому репозиторию. После этого мы сможем задействовать другие API и смоделировать связи между пользователями GitHub, следующими друг за другом и имеющими общие интересы.

Кроме того, мы познакомимся с некоторыми фундаментальными методами анализа графов, которые называют *мерами центральности*. Несмотря на очевидную пользу визуального представления графов, многие графы просто слишком велики или сложны, чтобы их можно было визуальным образом исследовать, поэтому для аналитической оценки аспектов структуры сети могут пригодиться меры центральности. (Но не волнуйтесь, прежде чем закончить эту главу, мы представим также методы визуализации графов.)

8.4.1. Начало создания графа интересов

Как уже говорилось, граф интересов и социальный граф² — не одно и то же. Если социальный граф, прежде всего, представляющий связи между людьми,

¹ <http://bit.ly/1a1ocFV>.

² <http://bit.ly/1a1ofl4>.

требует наличия взаимоотношений между вовлеченными сторонами, то граф интересов связывает людей и их интересы посредством однонаправленных связей (ребер). Эти два понятия не являются полностью непересекающимися понятиями, но не следует путать связь, соединяющую двух пользователей GitHub, с социальной связью, потому что она является связью «по интересам», поскольку не требует взаимного приятия.



Классическим примером гибридной графовой модели, которую можно назвать социальным графом интересов, является Facebook. Эта социальная сеть начиналась как технологическая платформа, основанная на понятии социального графа, но появление кнопки Like (Нравится) превратило эту платформу в гибридную, которую можно описать как социальный граф интересов. Она явно представляет связи между людьми, а также между людьми и интересующими их вещами. Социальная сеть Twitter всегда была разновидностью графа интересов с асимметричной моделью «следования», которую можно интерпретировать как связь между человеком и интересующими его сущностями (которые могут быть людьми).

В примерах 8.5 и 8.6 приводится код, конструирующий начальные отношения «интереса» между пользователями и репозиториями и демонстрирующий, как исследовать возникающую графовую структуру. Первоначально построенный граф можно назвать *эгографом*, в том смысле, что он имеет центральную точку (эго) — основу для большинства (в данном случае всех) ребер. Эгограф иногда называют «звездообразным графом» или «колесом», потому что он напоминает втулку колеса с исходящими из нее спицами или звезду.

С точки зрения строения граф содержит узлы двух типов и ребра одного типа, как показано на рис. 8.3.



Рис. 8.3. Базовая структура графа, включающая юзеров GitHub, заинтересовавшихся репозиторием



В качестве отправной точки мы используем схему, изображенную на рис. 8.3, и будем модифицировать ее по мере продвижения по оставшейся части главы.

Моделирование данных имеет ряд тонких, но важных ограничений, одним из которых является предотвращение конфликтов имен: имена пользователей и названия репозиториев могут совпадать (и часто совпадают) друг с другом. Например, в GitHub может иметься пользователь с именем «ptwobrussell», а также несколько репозиториев с названием «ptwobrussell». Напомню, что метод `add_edge` использует первые два параметра как уникальные идентификаторы, мы можем добавить в конец этих идентификаторов «(user)» или «(repo)», чтобы гарантировать уникальность всех узлов в графе. С точки зрения моделирования с помощью NetworkX добавление типа узла решает эту проблему в большинстве случаев.

Репозитории, принадлежащие разным пользователям, тоже могут иметь одинаковые названия и быть ответвлениями одного и того же базового репозитория или полностью отдельными проектами. На данный момент эта деталь нас не интересует, но как только мы начнем добавлять в граф другие репозитории, заинтересовавшие пользователей GitHub, вероятность таких конфликтов увеличится.

Допустить возможность таких конфликтов или реализовать такую стратегию конструирования графа, которая исключала бы их, — это архитектурное решение, влекущее определенные последствия. Например, иногда желательно, чтобы ответвления одного и того же репозитория свортывались в один узел графа и не рассматривались как самостоятельные репозитории, но уж точно не хотелось бы, чтобы совершенно разные проекты с одинаковыми названиями объединялись в один узел.



Учитывая ограниченность решаемой нами задачи, которая изначально фокусируется на конкретном репозитории, мы решили не обременять себя сложностями, связанными с совпадением названий репозиториев.

А теперь рассмотрим пример 8.5, конструирующий эгограф из репозитория и заинтересовавшихся им пользователей, и пример 8.6, представляющий несколько полезных операций с графами.

Пример 8.5. Конструирование эгографа из репозитория и его звездочетов

```
# Дополняет начальный граф ребрами (интересами), связывающими репозиторий
# с интересующимися им пользователями. Предпринимается попытка избежать
# конфликтов имен добавлением типов узлов к их идентификаторам.

import networkx as nx

g = nx.DiGraph()
g.add_node(repo.name + '(repo)', type='repo', lang=repo.language, owner=user.login)

for sg in stargazers:
    g.add_node(sg.login + '(user)', type='user')
    g.add_edge(sg.login + '(user)', repo.name + '(repo)', type='gazes')
```

Пример 8.6. Некоторые полезные операции с графами

Эксперименты с текущим графом, чтобы лучше понять, как работает NetworkX

```
print(nx.info(g))
print()
print(g.node['Mining-the-Social-Web(repo)'])
print(g.node['ptwobrussell(user)'])
print()
print(g['ptwobrussell(user)']['Mining-the-Social-Web(repo)'])

# Следующая строка возбудит исключение KeyError из-за отсутствия такого узла:
# print g['Mining-the-Social-Web(repo)']['ptwobrussell(user)']
print()
print(g['ptwobrussell(user)'])
print(g['Mining-the-Social-Web(repo)'])
print()
print(g.in_edges(['ptwobrussell(user)']))
print(g.out_edges(['ptwobrussell(user)']))
print()
print(g.in_edges(['Mining-the-Social-Web(repo)']))
print(g.out_edges(['Mining-the-Social-Web(repo)']))
```

Ниже приводится (сокращенный) вывод, демонстрирующий возможности только что показанных операций с графами:

```
Name:
Type: DiGraph
Number of nodes: 1117
Number of edges: 1116
Average in degree: 0.9991
Average out degree: 0.9991
```

```
{'lang': u'JavaScript', 'owner': u'ptwobrussell', 'type': 'repo'}
{'type': 'user'}
```

```
{'type': 'gazes'}

{u'Mining-the-Social-Web(repo)': {'type': 'gazes'}}
{}

[]
[('ptwobrussell(user)', u'Mining-the-Social-Web(repo)')]

[(u'gregmoreno(user)', 'Mining-the-Social-Web(repo)'),
 (u'SathishRaju(user)', 'Mining-the-Social-Web(repo)'),
 ...
]
[]
```

Получив начальный граф интересов, можно немного поразмышлять над выбором наиболее интересных направлений, куда можно двинуться дальше. На данный момент нам известно, что интерес к анализу данных из социальных сетей проявили 1117 пользователей, о чем свидетельствуют звезды, присвоенные ими репозиторию Mining-the-Social-Web пользователя ptwobrussell. Как и ожидалось, число ребер в графе на единицу меньше числа узлов. Причина этого в том, что на данный момент между репозиторием и пользователями, давшими звезду, существует прямая связь.

Как вы уже знаете, среднее степени входа и среднее степени выхода определяют нормализованную оценку плотности графа. Значение 0,9991 подтверждает это. Мы знаем, что в графе имеется 1117 узлов, соответствующих звездочетам, каждый из которых имеет степень выхода, равную 1, и один узел, соответствующий репозиторию, со степенью входа, равной 1117. Иначе говоря, мы знаем, что число ребер в графе на единицу меньше числа узлов. Граф имеет довольно низкую плотность, учитывая, что максимальное значение средней степени в этом случае равно 1117.

Было бы заманчиво поразмышлять о топологии графа, зная, что визуально его можно представить в виде звезды, и попытаться установить связь со значением 0,9991. У нас действительно есть один узел, связанный со всеми другими узлами в графе, но было бы ошибкой обобщать и пытаться установить связь со средним значением степени, приблизительно равным 1,0, опираясь на единственный узел. С тем же успехом связи между 1117 узлами можно перераспределить множеством других способов и получить то же значение 0,9991. Для поддержки этого вывода нам нужно рассмотреть дополнительные аналитические характеристики, такие как меры центральности, о которых рассказывается в следующем разделе.

8.4.2. Вычисление мер центральности графа

Мера центральности — это фундаментальная аналитическая характеристика графа, которая дает представление об относительной важности конкретного узла в графе. Рассмотрим далее следующие меры центральности, которые помогут нам тщательнее исследовать графы и получать более полное представление о сетях.

Центральность по степени (degree centrality)

Центральность по степени — это число ребер, связывающих узел графа с другими узлами. Эту меру центральности можно рассматривать как частотный анализ ребер узлов и использовать для оценки однородности, поиска узлов с наибольшим и наименьшим числом ребер или с целью выявления иных закономерностей, которые помогут получить более полное представление о топологии сети, основываясь на количестве связей. Центральность по степени — это лишь один из аспектов, которые можно использовать в рассуждениях о роли узла в сети, и ее определение обеспечивает хорошую начальную точку для выявления выбросов или аномалий в смысле числа связей с другими узлами в графе. Из предыдущей дискуссии мы также знаем, что среднее центральности по степени определяет общую плотность графа. В пакете NetworkX имеется встроенная функция `networkx.degree_centrality`, вычисляющая центральность по степени узлов в графе. Она возвращает словарь, отображающий идентификаторы узлов в соответствующие им значения центральности по степени.

Центральность по посредничеству (betweenness centrality)

Центральность по посредничеству определяет, как часто данный узел соединяет любые другие узлы в графе, то есть оказывается на пути *между* ними. Центральность по посредничеству можно рассматривать как меру важности узла как посредника или шлюза для соединения других узлов. Часто потеря узлов с высокой центральностью по посредничеству может отрицательно сказаться на потоке энергии¹ в графе, а иногда даже разрушить граф на меньшие подграфы. В пакете NetworkX имеется встроенная функция `networkx.betweenness_centrality`, вычисляющая центральность по посредничеству узлов в графе. Она возвращает словарь, отображающий

¹ Здесь термин «энергия» используется для обобщенного описания потоков в абстрактном графе.

идентификаторы узлов в соответствующие им значения центральности по посредничеству.

Центральность по близости (closeness centrality)

Центральность по близости — это мера, определяющая, насколько тесно связан («близок») узел со всеми остальными узлами в графе. Эта мера центральности также основывается на понятии кратчайших путей в графе и дает представление, насколько сильно интегрирован данный узел в граф. В отличие от центральности по посредничеству, сообщающей, насколько узел важен как промежуточное звено, соединяющее другие узлы, центральность по близости в большей мере учитывает прямые соединения. Близость можно рассматривать как способность узла распространять энергию во все другие узлы в графе. В пакете NetworkX имеется встроенная функция `networkx.closeness_centrality`, вычисляющая центральность по близости узлов в графе. Она возвращает словарь, отображающий идентификаторы узлов в соответствующие им значения центральности по близости.



Пакет NetworkX поддерживает вычисление нескольких мер центральности¹, описание которых можно найти в онлайн-документации.

На рис. 8.4 изображен граф Krackhardt kite², хорошо изученный в анализе социальных сетей, который иллюстрирует различия между мерами центральности, представленными в этом разделе. Свое название «kite» (бумажный змей) он получил благодаря сходству его визуального представления с бумажным змеем.

В примере 8.7 приводится код, который загружает этот граф из NetworkX и вычисляет для него меры центральности, которые показаны в табл. 8.1 ниже. Обратите внимание, что этот граф часто упоминается с отсылкой к социальным сетям. Ребра в графе неориентированы, потому что связи в социальной сети подразумевают их принятие с обеих сторон. В NetworkX такие неориентированные графы являются экземплярами `networkx.Graph`, а не `networkx.DiGraph`.

¹ <http://bit.ly/2MC1ZGV>.

² <http://bit.ly/1a1oixa>.

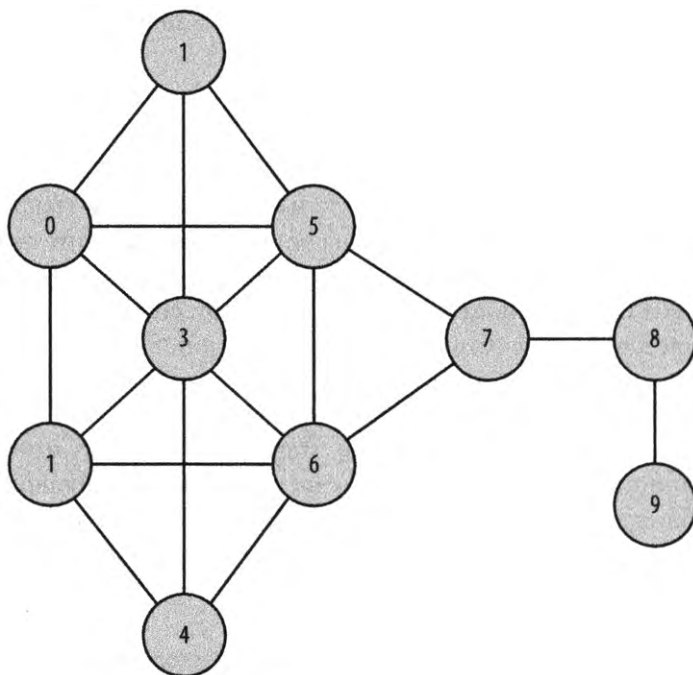


Рис. 8.4. Граф Krackhardt kite, иллюстрирующий меры центральности по степени, по посредничеству и по близости

Пример 8.7. Вычисление мер центральности по степени, по посредничеству и по близости для узлов графа Krackhardt kite

```
from operator import itemgetter
from IPython.display import HTML
from IPython.core.display import display

display(HTML(''))

# Классический граф Krackhardt kite
kkg = nx.generators.small.krackhardt_kite_graph()

print("Degree Centrality")
print(sorted(nx.degree_centrality(kkg).items(),
             key=itemgetter(1), reverse=True))
print()

print("Betweenness Centrality")
print(sorted(nx.betweenness_centrality(kkg).items(),
             key=itemgetter(1), reverse=True))
```

```
print()

print("Closeness Centrality")
print(sorted(nx.closeness_centrality(kkg).items(),
      key=itemgetter(1), reverse=True))
```

Таблица 8.1. Значения центральности по степени, по посредничеству и по близости для узлов графа Krackhardt kite (максимальные значения в каждой колонке выделены жирным шрифтом, чтобы вы легко могли проверить свои предположения относительно графа на рис. 8.4)

Узел	Центральность по степени	Центральность по посредничеству	Центральность по близости
0	0,44	0,02	0,53
1	0,44	0,02	0,53
2	0,33	0,00	0,50
3	0,67	0,10	0,60
4	0,33	0,00	0,50
5	0,55	0,20	0,64
6	0,55	0,20	0,64
7	0,33	0,39	0,60
8	0,22	0,22	0,43
9	0,11	0,00	0,31

Прежде чем переходить к следующему разделу, уделите немного времени для знакомства с графом Krackhardt kite и мерами центральности его узлов. Мы будем использовать эти меры далее в этой главе.

8.4.3. Расширение графа интересов ребрами «следования» между пользователями

Кроме присвоения звезд и ветвления репозитория, GitHub поддерживает также понятие «следования», заимствованное из Twitter. В этом разделе мы вновь обратимся к GitHub API и добавим эти отношения в наш граф. Из рассуждений о том, что Twitter по сути является графом интересов (одно из таких обсуждений можно найти, например, в разделе «Причины популярности Twitter» главы 1),

вы знаете, что эта социальная сеть в большей мере ориентирована на отношения заинтересованности, потому что отношение «следования» по большому счету является отношением «заинтересован в».

Можно побиться об заклад, что владелец репозитория пользуется популярностью в сообществе пользователей, давших звезду этому репозиторию, но кто еще может быть популярным в этом сообществе? Ответ на этот вопрос, безусловно, поможет расширить понимание и станет основой для дальнейшего анализа. Попробуем узнать этот ответ, обратившись к GitHub Followers API¹, чтобы получить списки последователей для каждого пользователя в графе, и добавим ребра, соответствующие отношениям следования. С точки зрения модели мы просто добавим в граф дополнительные ребра; никаких новых узлов при этом добавляться не будет.

Можно было бы добавить в граф все отношения следования, возвращаемые GitHub, но в данный момент мы ограничимся только пользователями, явно проявившими интерес к репозиторию, на основе которого был построен начальный граф. В примере 8.8 приводится фрагмент кода, который добавляет ребра следования в граф, а на рис. 8.5 изображена исправленная схема графа, включающая эти отношения.



Чтобы превысить ограничение в 5000 аутентифицированных запросов в час, нужно посылать больше 80 запросов в минуту. Это довольно сложно, учитывая задержку, возникающую в каждом запросе, поэтому в примерах кода для этой главы отсутствует специальная логика, помогающая предотвратить нарушение ограничений.

Пример 8.8. Добавление в граф интересов дополнительных ребер, соответствующих отношениям «следования»

```
# Добавляет (социальные) ребра, определяющие отношения следования между
# пользователями. На выполнение сценария может потребоваться довольно
# много времени из-за возможных задержек, возникающих в вызовах GitHub API.
# Примерное число запросов в каждой итерации цикла можно оценить как
# math.ceil(sg.get_followers() / 100.0) для случая, когда в ответ на один запрос
# возвращается до 100 элементов.
```

```
import sys
```

```
for i, sg in enumerate(stargazers):
```

¹ <http://bit.ly/1a1oixo>.

```
# Добавить ребра "следования" между пользователями в графе,
# если это отношение существует
try:
    for follower in sg.get_followers():
        if follower.login + '(user)' in g:
            g.add_edge(follower.login + '(user)', sg.login + '(user)',
                       type='follows')
except Exception as e: #ssl.SSLError
    print("Encountered an error fetching followers for", sg.login,
          "Skipping.", file=sys.stderr)
    print(e, file=sys.stderr)

print("Processed", i+1, " stargazers. Num nodes/edges in graph",
      g.number_of_nodes(), "/", g.number_of_edges())
print("Rate limit remaining", client.rate_limiting)
```

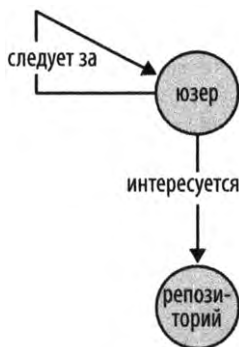


Рис. 8.5. Базовая структура графа, включающая юзеров GitHub, заинтересовавшихся репозиторием, и связи следования между ними

С включением в граф дополнительных данных о заинтересованности мы получаем более широкие возможности для анализа. Теперь можно выполнить обход графа и вычислить оценку популярности пользователей, подсчитав для каждого число входящих ребер «следует за», как показано в примере 8.9. Что самое интересное, этот шаг позволяет нам быстро определить, кто из наиболее интересных или влиятельных пользователей занимается исследованиями в этой области.

Пример 8.9. Исследование графа с вновь добавленными ребрами «следует за»

```
from operator import itemgetter
from collections import Counter
```

```

# Показать, сколько социальных ребер добавлено на предыдущем шаге
print(nx.info(g))
print()

# Число ребер "следует за" будет меньше
print(len([e for e in g.edges_iter(data=True) if e[2]['type'] == 'follows']))
print()

# Возможно, самым популярным в этом графе является владелец репозитория
print(len([e
    for e in g.edges_iter(data=True)
    if e[2]['type'] == 'follows' and e[1] == 'ptwobrussell(user)']))
print()

# Узнаем число смежных ребер для каждого узла
print(sorted([n for n in g.degree_iter()], key=itemgetter(1), reverse=True)[:10])
print()

# Найдем отношение входящих и исходящих ребер для пары пользователей
# с наибольшей центральностью по степени...

# Пользователь, следующий за многими, но за которым почти никто не следует

print(len(g.out_edges('mcanthony(user)')))
print(len(g.in_edges('mcanthony(user)')))
print()

# Пользователь, за которым следуют многие, но сам он ни за кем не следует

print(len(g.out_edges('ptwobrussell(user)')))
print(len(g.in_edges('ptwobrussell(user)')))
print()

c = Counter([e[1] for e in g.edges_iter(data=True) if e[2]['type'] == 'follows'])
popular_users = [ (u, f) for (u, f) in c.most_common() if f > 1 ]
print("Number of popular users", len(popular_users))
print("Top 10 popular users:", popular_users[:10])

```

Учитывая, что основой для создания графа стал репозиторий Mining-the-Social-Web, правдоподобной выглядит гипотеза, что пользователи, интересующиеся этой темой, могут иметь некоторое отношение или интерес к анализу данных вообще и, может быть, к языку программирования Python, поскольку код в обсуждаемом репозитории написан на Python. Давайте посмотрим, имеют ли какую-то связь с языком Python наиболее популярные пользователи, выявленные в примере 8.9.

Вот вывод, полученный кодом в этом примере:

```
Name:
Type: DiGraph
Number of nodes: 1117
Number of edges: 2721
Average in degree: 2.4360
Average out degree: 2.4360

1605

125

[('Mining-the-Social-Web(repo)', 1116),
 ('angusshire(user)', 511),
 ('kennethreitz(user)', 156),
 ('ptwobrussell(user)', 126),
 ('VagrantStory(user)', 106),
 ('beali(user)', 92),
 ('trietptm(user)', 71),
 ('rohithadassanayake(user)', 48),
 ('mcanthony(user)', 37),
 ('daimajia(user)', 36)]

32
5

1

125

Number of popular users 270
Top 10 popular users:
[('kennethreitz(user)', 153),
 ('ptwobrussell(user)', 125),
 ('daimajia(user)', 32),
 ('hammer(user)', 21),
 ('isnowfy(user)', 20),
 ('jakubroztocil(user)', 20),
 ('japerk(user)', 19),
 ('angusshire(user)', 18),
 ('dgryski(user)', 12),
 ('tswicegood(user)', 11)]
```

Как и ожидалось, владелец репозитория, ставшего основой графа интересов, — пользователь `ptwobrussell`¹, — оказался в числе самых популярных пользова-

¹ <http://bit.ly/1a1o4GC>.

телей в этом графе. Но в топ-10 есть другой, более популярный пользователь (kennethreitz), имеющий 153 последователя, а также еще несколько пользователей с большим числом последователей. Кроме всего прочего, выяснилось, что kennethreitz¹ является автором популярного пакета requests для Python, использовавшегося в этой книге. Также можно отметить пользователя mcanthony, который следует за многими, но за которым следует очень небольшое число других пользователей. (Мы еще вернемся к этому наблюдению чуть ниже.)

Применение мер центральности

Прежде чем продолжить работу, сохраним представление нашего графа, чтобы иметь стабильный срез текущего состояния на случай, если нам потребуется поэкспериментировать с графом и затем восстановить его или мы захотим передать его кому-нибудь еще. Пример 8.10 демонстрирует, как сохранить и восстановить граф с помощью инструментов архивирования, встроенных в NetworkX.

Пример 8.10. Сохранение (архивирование) графа на диск

```
# Сохранить (архивировать) граф на диск
nx.write_gpickle(g, "resources/ch08-github/data/github.gpickle.1")

# А вот так можно загрузить граф с диска...
# import networkx as nx
# g = nx.read_gpickle("resources/ch08-github/data/github.gpickle.1")
```

Сохранив резервную копию на диск, вычислим теперь меры центральности, как было описано в предыдущем разделе, и попробуем интерпретировать результаты. Поскольку известно, что Mining-the-Social-Web(repo) является *суперузлом* в графе и связывает подавляющее большинство пользователей (в данном случае всех), удалим его из графа, чтобы лучше понять динамику сети. После этого в графе останутся только ребра «следует за». Код в примере 8.11 реализует отправную точку для дальнейшего анализа.

Пример 8.11. Вычисление мер центральности для узлов графа интересов

```
from operator import itemgetter

# Создать копию графа, чтобы иметь возможность последовательно
# изменять копию в ходе экспериментов
```

¹ <http://bit.ly/1a1ojkT>.

```

h = g.cory()

# Удалить начальный узел графа интересов, являющийся суперузлом, чтобы
# получить более полное представление о динамике сети

h.remove_node('Mining-the-Social-Web(repo)')

# Удалить все другие узлы, которые выглядят как суперузлы.
# Выявить такие узлы можно по пороговому значению или
# некоторой эвристике, полученной по результатам исследования.

# Вывести меры центральности для топ-10 узлов

dc = sorted(nx.degree_centrality(h).items(),
            key=itemgetter(1), reverse=True)

print("Degree Centrality")
print(dc[:10])
print()

bc = sorted(nx.betweenness_centrality(h).items(),
            key=itemgetter(1), reverse=True)

print("Betweenness Centrality")
print(bc[:10])
print()

cc = sorted(nx.closeness_centrality(h).items(),
            key=itemgetter(1), reverse=True)
print(cc[:10])

```

Далее приводятся полученные результаты:

```

Degree Centrality
[('angusshire(user)', 0.45739910313901344),
 ('kennethreitz(user)', 0.13901345291479822),
 ('ptwobrussell(user)', 0.11210762331838565),
 ('VagrantStory(user)', 0.09417040358744394),
 ('beali(user)', 0.08161434977578476),
 ('trietptm(user)', 0.06278026905829596),
 ('rohithadassanayake(user)', 0.042152466367713005),
 ('mcanthony(user)', 0.03228699551569507),
 ('daimajia(user)', 0.03139013452914798),
 ('JTSO(user)', 0.029596412556053813)]

```

```

Betweenness Centrality
[('angusshire(user)', 0.012199321617913778),
 ('rohithadassanayake(user)', 0.0024989064307240636),

```



```
('trietptm(user)', 0.0016462150915044311),
('douglas(user)', 0.0014378758725072656),
('JT5D(user)', 0.0006630082719888302),
('mcanthony(user)', 0.0006042022778087548),
('VagrantStory(user)', 0.0005563053609377326),
('beali(user)', 0.0005419295788331876),
('LawrencePeng(user)', 0.0005133545798221231),
('frac(user)', 0.0004898921995636457)]
```

Closeness Centrality

```
[('angusshire(user)', 0.45124556968457424),
('VagrantStory(user)', 0.2824285214515154),
('beali(user)', 0.2801929394875192),
('trietptm(user)', 0.2665936169015141),
('rohithadassanayake(user)', 0.26460080747284836),
('mcanthony(user)', 0.255887045941614),
('marctmiller(user)', 0.2522401996811634),
('cwz8202(user)', 0.24927963395720612),
('uetchy(user)', 0.24792169042592171),
('LawrencePeng(user)', 0.24734423307244519)]
```

Как и ожидалось, пользователи ptwobrussell и kennethreitz оказались близко к началу списка топ-10 пользователей с наибольшей центральностью по степени. Однако на вершине всех трех чартов оказался другой пользователь, angusshire. Этот пользователь является суперузлом, который сам следует и за которым следуют многие тысячи других пользователей. Если убрать этого пользователя из графа, это почти наверняка изменит динамику сети.

Еще одно наблюдение, которое хотелось бы отметить, — значения центральности по близости и по степени намного выше значений центральности по посредничеству, которые в свою очередь очень близки к нулю. В контексте отношений «следования» это означает, что в графе нет ни одного пользователя, который выступал бы в роли моста, связывающего других пользователей. Это вполне объяснимо, потому что изначально граф был создан на основе репозитория, являющегося общим интересом. Мы могли бы ожидать обнаружить некоторого пользователя с высокой оценкой посредничества, но нет ничего неожиданного и в том, что это не так. Если бы в качестве основы графа интересов был выбран пользователь, тогда динамика могла бы оказаться совсем иной.

Наконец, обратите внимание, что несмотря на высокую популярность пользователей ptwobrussell и kennethreitz, они не попали в топ-10 пользователей с наибольшими значениями центральности по близости. Зато в этом списке

появилось несколько других пользователей, имеющих значительные оценки близости, и было бы интересно исследовать их. Имейте в виду, что динамика сети будет отличаться для разных сообществ.



Было бы интересно сравнить динамику сети двух разных сообществ, таких как сообщества Ruby on Rails и Django. Можете также попробовать сравнить динамику сообществ сторонников Microsoft и Linux.

Добавление других репозиторий в граф интересов

В общем и целом наш анализ ребер «следует за» в графе не выявил ничего интересного, что не так уж удивительно, если вспомнить, что за основу графа интересов был выбран репозиторий, привлечший разрозненных пользователей со всего мира. Следующим шагом, достойным внимания, могла бы стать попытка выяснить дополнительные интересы каждого пользователя в графе и добавить в граф репозитории, которые они отметили звездами. Это поможет нам ответить на два важных вопроса: какие другие репозитории привлекают членов сообщества, объединенного интересом к анализу данных из социальных сетей (и в меньшей степени интересом к языку Python), и какие языки программирования популярны в сообществе, воспользовавшись тем фактом, что GitHub пытается индексировать репозитории и определять используемые языки программирования.

Процесс добавления в граф репозиторий и ребер «интересуется» является простым дополнением к работе, проделанной выше в этой главе. GitHub API «List repositories being starred»¹ (список репозиторий, получивших звезды от пользователя) позволяет легко получить список репозиторий, которые данный пользователь отметил звездой, нам останется только выполнить обход полученных результатов и добавить в граф соответствующие узлы и ребра, как мы это уже делали. В примере 8.12 приводится код, решающий эту задачу. Он добавляет в граф значительный объем данных и для его выполнения может потребоваться значительное время. Если вы работаете с репозиторием, имеющим больше нескольких десятков звезд, вам придется подождать.

Пример 8.12. Добавление репозиторий, отмеченных пользователями из графа

```
# Выполнить обход звездочетов и добавить в граф другие репозитории,  
# отмеченные ими, и соответствующие ребра для поиска дополнительных интересов
```

¹ <http://bit.ly/1a1oc8X>.

```
MAX_REPOS = 500
```

```
for i, sg in enumerate(stargazers):
    print(sg.login)
    try:
        for starred in sg.get_starred()[MAX_REPOS]: # Slice to avoid supernodes
            g.add_node(starred.name + '(repo)', type='repo', lang=starred.language,
                       owner=starred.owner.login)
            g.add_edge(sg.login + '(user)', starred.name + '(repo)', type='gazes')
        except Exception as e: #ssl.SSLError:
            print("Encountered an error fetching starred repos for", sg.login,
                  "Skipping.")

print("Processed", i+1, "stargazers' starred repos")
print("Num nodes/edges in graph", g.number_of_nodes(), "/", g.number_of_edges())
print("Rate limit", client.rate_limiting)
```

При конструировании этого графа возникает одна малозаметная проблема: несмотря на то что большинство пользователей отметили «разумное» число репозиторий, некоторые раздали очень большое число звезд, выйдя за рамки статистической нормы и внося в граф непропорционально большое количество ребер. Как отмечалось выше, узел с экстремально большим числом ребер и образующий аномалию с большим запасом, называется *суперузлом*. Часто нежелательно иметь в графе (особенно в графе NetworkX, который целиком хранится в оперативной памяти) суперузлы, потому что в лучшем случае они затрудняют обход графа, а в худшем могут вызвать ошибку нехватки памяти. Целесообразность включения или невключения суперузлов должна определяться конкретными целями и задачами.

Разумным решением, которое мы использовали в примере 8.12, чтобы предотвратить включение в граф суперузлов, является простое ограничение числа репозиторий, которое можно добавить для одного пользователя. В данном конкретном случае мы установили довольно высокое ограничение (500 репозиторий), обрезая результаты для обхода в цикле `for` как `get_starred()[ : 500]`. Позднее, если мы решим выполнить полный обход суперузлов, нам останется только выбрать из графа узлы с большим числом исходящих ребер.



По мере добавления новых данных в граф Python, в том числе и серверное ядро Jupyter Notebook, будет использовать столько памяти, сколько понадобится. Если попытаться создать граф настолько большой, что операционная система не сможет функционировать, процесс супервизора может принудительно завершить процесс Python, потребляющий слишком много памяти.

Теперь, получив граф с дополнительными репозиториями, можно заняться куда более интересным делом — выполнением запросов к графу. Кроме вычисления простых статистик мы можем задать множество вопросов и получить на них ответы. Например, было бы интересно выявить пользователя, владеющего самым большим числом наблюдаемых репозиториях. Еще один насущный вопрос — какие еще имеются в графе репозитории, пользующиеся большой популярностью, кроме исходного репозитория, на основе которого был создан граф. В примере 8.13 вы найдете блок кода, отвечающий на этот вопрос и обозначающий отправную точку для дальнейшего анализа.



Метод `get_starred` из пакета `PyGithub`, обертка вокруг GitHub API «List repositories being starred»¹ (список репозиториях, получивших звезды от пользователя), возвращает несколько других интересных свойств, которые могут вас заинтересовать в будущих экспериментах. Обязательно ознакомьтесь с документацией, описывающей API, чтобы не пропустить ничего, что могло бы пригодиться вам в исследовании этой области.

Пример 8.13. Исследование графа после добавления дополнительных репозиториях

Эксперименты с пользователями/репозиториями в графе

```
from operator import itemgetter
```

```
print(nx.info(g))
```

```
print()
```

Получить список репозиториях из графа

```
repos = [n for n in g.nodes_iter() if g.node[n]['type'] == 'repo']
```

Наиболее популярные репозиториях

```
print("Popular repositories")
```

```
print(sorted([(n,d)
```

```
    for (n,d) in g.in_degree_iter()
```

```
        if g.node[n]['type'] == 'repo'],
```

```
    key=itemgetter(1), reverse=True)[:10])
```

```
print()
```

Проекты, отмеченные пользователем

```
print("Respositories that ptwobrussell has bookmarked")
```

```
print([(n,g.node[n]['lang'])
```

```
    for n in g['ptwobrussell(user)']
```

¹ <http://bit.ly/1a1oc8X>.

```

        if g['ptwobrussell(user)'][n]['type'] == 'gazes'])
print()

# Языки программирования для каждого пользователя

print("Programming languages ptwobrussell is interested in")
print(list(set([g.node[n]['lang']
                for n in g['ptwobrussell(user)']
                if g['ptwobrussell(user)'][n]['type'] == 'gazes'])))

print()

# Найти суперузлы в графе по большому числу исходящих узлов

print("Supernode candidates")
print(sorted([(n, len(g.out_edges(n)))
              for n in g.nodes_iter()
              if g.node[n]['type'] == 'user' and len(g.out_edges(n)) > 500],
            key=itemgetter(1), reverse=True))

```

Пример вывода:

```

Name:
Type: DiGraph
Number of nodes: 106643
Number of edges: 383807
Average in degree: 3.5990
Average out degree: 3.5990

Popular repositories
[('Mining-the-Social-Web(repo)', 1116),
 ('bootstrap(repo)', 246),
 ('d3(repo)', 224),
 ('tensorflow(repo)', 204),
 ('dotfiles(repo)', 196),
 ('free-programming-books(repo)', 179),
 ('Mining-the-Social-Web-2nd-Edition(repo)', 147),
 ('requests(repo)', 138),
 ('storm(repo)', 137),
 ('Probabilistic-Programming-and-Bayesian-Methods-for-Hackers(repo)', 136)]

Respositories that ptwobrussell has bookmarked
[('Mining-the-Social-Web(repo)', 'JavaScript'),
 ('re2(repo)', 'C++'),
 ('google-cloud-python(repo)', 'Python'),
 ('CNTK(repo)', 'C++'),
 ('django-s3direct(repo)', 'Python'),
 ('medium-editor-insert-plugin(repo)', 'JavaScript'),
 ('django-ckeditor(repo)', 'JavaScript'),
 ('rq(repo)', 'Python'),

```

```

('x-editable(repo)', 'JavaScript'),
...
]

Programming languages ptwobrussell is interested in
['Python', 'HTML', 'JavaScript', 'Ruby', 'CSS', 'Common Lisp',
 'CoffeeScript', 'Objective-C', 'PostScript',
 'Jupyter Notebook', 'Perl', 'C#', 'C', 'C++', 'Lua', 'Java',
 None, 'Go', 'Shell', 'Clojure']

Supernode candidates
[('angusshire(user)', 1004),
 ('VagrantStory(user)', 618),
 ('beali(user)', 605),
 ('trietptm(user)', 586),
 ('rohithadassanayake(user)', 579),
 ('zmughal(user)', 556),
 ('LJ001(user)', 556),
 ('JTS0(user)', 554),
 ('kcnickerson(user)', 549),
 ...
]
```

Первое, что сразу бросается в глаза: число ребер в новом графе на три порядка больше, чем в предыдущем, а число узлов увеличилось больше, чем на порядок. Благодаря более сложной динамике сети анализ становится особенно интересным. Однако увеличение сложности динамики влечет значительное увеличение времени, необходимого пакету NetworkX для вычисления глобальных статистик. Имейте в виду: тот факт, что граф целиком находится в оперативной памяти, не означает, что все вычисления будут производиться быстро. В этих обстоятельствах вам может пригодиться знание некоторых базовых принципов вычислений.

Вычислительные проблемы



Этот короткий раздел содержит более подробное объяснение некоторых математических сложностей, возникающих в графовых алгоритмах вычислений. Можете прочитать его прямо сейчас или вернуться к нему позже, если вы читаете эту главу в первый раз.

Из трех вычисляемых мер центральности самой простой, как известно, является мера центральности по степени, и она должна вычисляться относительно быстро, потому что для этого требуется только один раз обойти узлы графа

и подсчитать число соответствующих им ребер. Однако две другие меры центральности, по посредничеству и по близости, требуют вычисления минимального остовного дерева¹ (minimum spanning tree). Алгоритм минимального остовного дерева в пакете NetworkX² использует для этого алгоритм Краскала³ (Kruskal), который широко используется в учебных курсах по информатике. С точки зрения вычислительной сложности он имеет порядок $O(E \log E)$, где E представляет число ребер в графе. Обычно алгоритмы с такой сложностью считаются эффективными, но $100\,000 \times \log(100\,000)$ примерно равно одному миллиону операций, поэтому для выполнения анализа потребуется некоторое время.

Удаление суперузлов имеет решающее значение для достижения приемлемого времени выполнения сетевых алгоритмов, поэтому для проведения более детального анализа можно подумать о возможности выделения подграфов. Например, можно выборочно удалить из графа пользователей, основываясь на таких критериях, как число последователей, которое может служить свидетельством их важности для всей сети. Можно также подумать о введении порога с минимальным числом звезд для включения репозиториев.

При проведении анализа на больших графах рекомендуется исследовать меры центральности по отдельности, чтобы можно было быстро получать и просматривать результаты. Также для ускорения вычислений важно удалить из графа суперузлы, которые легко могут оказывать доминирующее влияние на сетевые алгоритмы. В зависимости от размера графа иногда бывает полезно увеличить объем памяти, доступной виртуальной машине.

8.4.4. Использование узлов в качестве точек опоры для увеличения эффективности запросов

Другой интересной характеристикой, достойной внимания, является популярность языков программирования, применяемых пользователями. Обычно пользователи отмечают проекты, реализованные на языках программирования, которые они (пользователи) хотя бы немного знают и могут использовать. У нас есть инструменты и данные для анализа пользователей и популярности языков программирования в нашем существующем графе, но в данный момент наша схема имеет один недостаток. Поскольку язык программирования моделируется

¹ <http://bit.ly/1a1omgr>.

² <http://bit.ly/2DdURBx>.

³ <http://bit.ly/1a1on3X>.

как атрибут репозитория, для ответа на нетривиальные вопросы нам придется просканировать все узлы репозитория и либо извлекать их, либо фильтровать по этому атрибуту.

Например, чтобы с текущей схемой узнать, на каких языках программирует некоторый пользователь, мы должны будем просмотреть все репозитории, отмеченные этим пользователем, извлечь значения свойства `lang` и вычислить частоты. Это решение не выглядит слишком громоздким, но представьте, что мы захотели узнать, сколько пользователей программирует на конкретном языке. Несмотря на то что текущая схема позволяет найти ответ, нам придется просканировать все узлы репозитория и подсчитать все входящие ребра «интересуется». Однако немного изменив схему графа, мы легко сможем получить искомый ответ, обратившись к единственному узлу. Это изменение включает добавление в граф отдельного узла для каждого языка программирования, имеющего входящие ребра `programs` («программирует») от пользователей, программирующих на этом языке, и исходящие ребра `implements` («реализован») к репозиториям.

На рис. 8.6 изображена окончательная схема графа, включающая языки программирования, а также ребра, связывающие пользователей, репозитории и языки программирования. Суть изменений заключается в том, что мы берем свойство узла и создаем в графе явное отношение, которое прежде было неявным. С точки зрения полноты у нас не появилось новых данных, но теперь мы можем обрабатывать некоторые запросы намного эффективнее. Несмотря на простоту схемы, вселенная возможных графов, привязанных к ней, которые с легкостью можно создавать и добывать ценные знания, огромна.

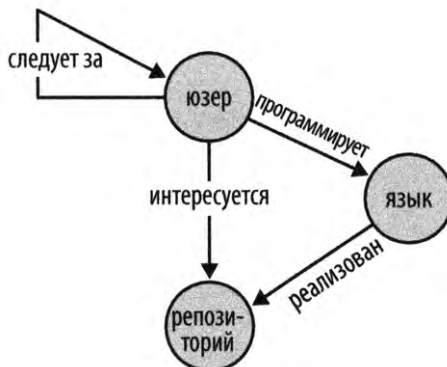


Рис. 8.6. Структура графа, включающая юзеров GitHub, репозитории и языки программирования

В примере 8.14 представлен код, преобразующий граф согласно окончательной схеме. Так как вся информация, необходимая для конструирования дополнительных узлов и ребер, уже имеется в графе (как вы помните, языки программирования уже хранятся в свойствах узлов репозитория), выполнять запросы к GitHub API не требуется.

Здорово иметь в графе по отдельному узлу для каждого языка программирования вместо свойств, разбросанных по всему графу, потому что единственный узел может выступать в роли естественной точки агрегирования. Наличие центральных точек агрегирования может сильно упростить обработку многих видов запросов, таких как поиск максимальных группировок (клик) в графе. Например, поиск максимальной клики пользователей, следующих за кем-то и программирующих на определенном языке, можно выполнить очень эффективно с помощью алгоритмов обнаружения клик¹ в NetworkX, поскольку требование конкретного узла языка программирования в клике значительно ограничивает область поиска.

Пример 8.14. Преобразование графа включением узлов с языками программирования

```
# Выполнить обход всех репозиториях и добавить ребра, связывающие
# пользователей с языками программирования. Также добавить ребра,
# связывающие языки программирования с репозиториями, чтобы получить
# хорошую "точку опоры".
```

```
repos = [n
    for n in g.nodes_iter()
    if g.node[n]['type'] == 'repo']

for repo in repos:
    lang = (g.node[repo]['lang'] or "") + "(lang)"

    stargazers = [u
        for (u, r, d) in g.in_edges_iter(repo, data=True)
        if d['type'] == 'gazes'
    ]

    for sg in stargazers:
        g.add_node(lang, type='lang')
        g.add_edge(sg, lang, type='programs')
        g.add_edge(lang, repo, type='implements')
```

¹ <http://bit.ly/2GDgBI6>.

Наша окончательная схема графа позволяет отвечать на самые разные вопросы. Вот некоторые из таких вопросов, на которые мы можем ответить прямо сейчас:

- На каких языках программируют конкретные пользователи?
- Сколько пользователей программирует на конкретном языке?
- Какие пользователи программируют на нескольких языках, таких как Python и JavaScript?
- Кто из программистов использует больше всего языков?
- Существует ли тесная связь между конкретными языками? (Например, если программист пишет на Python, можно ли сказать, опираясь на данные в графе, что тот же программист пишет также на JavaScript или Go?)

В примере 8.15 приводится фрагмент кода, отвечающий на многие из этих вопросов и некоторые другие, подобные им.

Пример 8.15. Примеры запросов к окончательному графу

```
# Некоторые статистики
```

```
print(nx.info(g))
print()
```

```
# Какие языки присутствуют в графе?
```

```
print([n
      for n in g.nodes_iter()
        if g.node[n]['type'] == 'lang'])
print()
```

```
# На каких языках программирует выбранный пользователь?
```

```
print([n
      for n in g['ptwobrussell(user)']
        if g['ptwobrussell(user)'][n]['type'] == 'programs'])
```

```
print()
```

```
# Какие языки программирования самые популярные?
```

```
print("Most popular languages")
print(sorted([(n, g.in_degree(n))
              for n in g.nodes_iter()]))
```

```

    if g.node[n]['type'] == 'lang'], key=itemgetter(1), reverse=True)[:10])
print()

# Сколько пользователей программирует на выбранном языке?
python_programmers = [u
    for (u, l) in g.in_edges_iter('Python(lang)')
    if g.node[u]['type'] == 'user']
print("Number of Python programmers:", len(python_programmers))
print()

javascript_programmers = [u for
    (u, l) in g.in_edges_iter('JavaScript(lang)')
    if g.node[u]['type'] == 'user']
print("Number of JavaScript programmers:", len(javascript_programmers))
print()

# Кто из пользователей программирует на обоих языках, Python и JavaScript?
print("Number of programmers who use JavaScript and Python")
print(len(set(python_programmers).intersection(set(javascript_programmers))))

# Программисты, использующие JavaScript, но не использующие Python
print("Number of programmers who use JavaScript but not Python")
print(len(set(javascript_programmers).difference(set(python_programmers))))

# Попробуйте самостоятельно определить, кто из программистов использует
# больше всего языков.

```

Пример вывода:

```

Name:
Type: DiGraph
Number of nodes: 106643
Number of edges: 383807
Average in degree: 3.5990
Average out degree: 3.5990

['JavaScript(lang)', 'Python(lang)', '(lang)', 'Shell(lang)', 'Go(lang)',
'C++(lang)', 'HTML(lang)', 'Scala(lang)', 'Objective-C(lang)',
'TypeScript(lang)', 'Java(lang)', 'C(lang)', 'Jupyter Notebook(lang)',
'CSS(lang)', 'Ruby(lang)', 'C#(lang)', 'Groovy(lang)', 'XSLT(lang)',
'Eagle(lang)', 'PostScript(lang)', 'R(lang)', 'PHP(lang)', 'Erlang(lang)',
'Elixir(lang)', 'CoffeeScript(lang)', 'Matlab(lang)', 'TeX(lang)',
'VimL(lang)', 'Haskell(lang)', 'Clojure(lang)', 'Makefile(lang)',
'Emacs Lisp(lang)', 'OCaml(lang)', 'Perl(lang)', 'Swift(lang)', 'Lua(lang)',
'COBOL(lang)', 'Batchfile(lang)', 'Visual Basic(lang)',
'Protocol Buffer(lang)', 'Assembly(lang)', 'Arduino(lang)', 'Cuda(lang)',
'Ada(lang)', 'Rust(lang)', 'HCL(lang)', 'Common Lisp(lang)',
'Objective-C++(lang)', 'GLSL(lang)', 'D(lang)', 'Dart(lang)',
'Standard ML(lang)', 'Vim script(lang)', 'Coq(lang)', 'FORTRAN(lang)',

```

```

'Julia(lang)', 'OpenSCAD(lang)', 'Kotlin(lang)', 'Pascal(lang)',
'Logos(lang)', 'Lean(lang)', 'Vue(lang)', 'Elm(lang)', 'Crystal(lang)',
'PowerShell(lang)', 'AppleScript(lang)', 'Scheme(lang)', 'Smarty(lang)',
'PLpgSQL(lang)', 'Groff(lang)', 'Lex(lang)', 'Cirru(lang)',
'Mathematica(lang)', 'BitBake(lang)', 'Fortran(lang)',
'DIGITAL Command Language(lang)', 'ActionScript(lang)', 'Smalltalk(lang)',
'Bro(lang)', 'Racket(lang)', 'Frege(lang)', 'POV-Ray SDL(lang)', 'M(lang)',
'Puppet(lang)', 'GAP(lang)', 'VHDL(lang)', 'Gherkin(lang)',
'Objective-J(lang)', 'Roff(lang)', 'VCL(lang)', 'Hack(lang)',
'MoonScript(lang)', 'Tcl(lang)', 'CMake(lang)', 'Yacc(lang)', 'Vala(lang)',
'ApacheConf(lang)', 'PigLatin(lang)', 'SMT(lang)',
'GCC Machine Description(lang)', 'F#(lang)', 'QML(lang)', 'Monkey(lang)',
'Processing(lang)', 'Parrot(lang)', 'Nix(lang)', 'Nginx(lang)',
'Nimrod(lang)', 'SQLPL(lang)', 'Web Ontology Language(lang)', 'Nu(lang)',
'Arc(lang)', 'Rascal(lang)', "Cap'n Proto(lang)", 'Gosu(lang)', 'NSIS(lang)',
'MTML(lang)', 'ColdFusion(lang)', 'LiveScript(lang)', 'Hy(lang)',
'OpenEdge ABL(lang)', 'KiCad(lang)', 'Perl6(lang)', 'Prolog(lang)',
'XQuery(lang)', 'AutoIt(lang)', 'LOLCODE(lang)', 'Verilog(lang)',
'NewLisp(lang)', 'Cucumber(lang)', 'PureScript(lang)', 'Awk(lang)',
'RAML(lang)', 'Haxe(lang)', 'Thrift(lang)', 'XML(lang)', 'SaltStack(lang)',
'Pure Data(lang)', 'SuperCollider(lang)', 'HaXe(lang)',
'Ragel in Ruby Host(lang)', 'API Blueprint(lang)', 'Squirrel(lang)',
'Red(lang)', 'NetLogo(lang)', 'Factor(lang)', 'CartoCSS(lang)', 'Rebol(lang)',
'REALbasic(lang)', 'Max(lang)', 'Chuck(lang)', 'AutoHotkey(lang)',
'Apex(lang)', 'ASP(lang)', 'Stata(lang)', 'nesC(lang)',
'Gettext Catalog(lang)', 'Modelica(lang)', 'Augeas(lang)', 'Inform 7(lang)',
'APL(lang)', 'LilyPond(lang)', 'Terra(lang)', 'IDL(lang)', 'Brainfuck(lang)',
'Idris(lang)', 'AspectJ(lang)', 'Opa(lang)', 'Nim(lang)', 'SQL(lang)',
'Ragel(lang)', 'M4(lang)', 'Grammatical Framework(lang)', 'Nemerle(lang)',
'AGS Script(lang)', 'MQL4(lang)', 'Smali(lang)', 'Pony(lang)', 'ANTLR(lang)',
'Handlebars(lang)', 'PLSQL(lang)', 'SAS(lang)', 'FreeMarker(lang)',
'Fancy(lang)', 'DM(lang)', 'Agda(lang)', 'Io(lang)', 'Limbo(lang)',
'Liquid(lang)', 'Gnuplot(lang)', 'Xtend(lang)', 'LLVM(lang)',
'BlitzBasic(lang)', 'TLA(lang)', 'Metal(lang)', 'Inno Setup(lang)',
'Diff(lang)', 'SRecode Template(lang)', 'Forth(lang)', 'SQF(lang)',
'PureBasic(lang)', 'Mirah(lang)', 'Bison(lang)', 'Oz(lang)',
'Game Maker Language(lang)', 'ABAP(lang)', 'Isabelle(lang)', 'AMPL(lang)',
'E(lang)', 'Ceylon(lang)', 'WebIDL(lang)', 'GDScript(lang)', 'Stan(lang)',
'Eiffel(lang)', 'Mercury(lang)', 'Delphi(lang)', 'Brightscript(lang)',
'Propeller Spin(lang)', 'Self(lang)', 'HLSL(lang)']

['JavaScript(lang)', 'C++(lang)', 'Java(lang)', 'PostScript(lang)',
'Python(lang)', 'HTML(lang)', 'Ruby(lang)', 'Go(lang)', 'C(lang)', '(lang)',
'Objective-C(lang)', 'Jupyter Notebook(lang)', 'CSS(lang)', 'Shell(lang)',
'Clojure(lang)', 'CoffeeScript(lang)', 'Lua(lang)', 'Perl(lang)',
'C#(lang)', 'Common Lisp(lang)']

```

Most popular languages

```
['JavaScript(lang)', 1115], ('Python(lang)', 1013), ('(lang)', 978),
```

```
('Java(lang)', 890), ('HTML(lang)', 873), ('Ruby(lang)', 870),  
('C++(lang)', 866), ('C(lang)', 841), ('Shell(lang)', 764),  
('CSS(lang)', 762)]
```

Number of Python programmers: 1013

Number of JavaScript programmers: 1115

Number of programmers who use JavaScript and Python

1013

Number of programmers who use JavaScript but not Python

102

Несмотря на концептуальную простоту схемы графа, добавление узлов с языками программирования увеличило число ребер почти на 50%! Как можно заметить из результатов некоторых запросов, пользователи программируют на большом количестве языков, и наибольшей популярностью пользуются JavaScript и Python. Исходный код примеров в оригинальном репозитории написан на Python, поэтому высокая популярность JavaScript среди пользователей может свидетельствовать, что они причастны к веб-разработке. При этом, конечно же, не следует забывать, что JavaScript сам является популярным языком, и часто существует высокая корреляция между JavaScript, как языком клиентских сценариев, и Python, как языком серверных сценариев. Появление '(lang)' как третьего по популярности языка может служить признаком, что 642 репозитория в GitHub, попавшие в эту категорию, не связаны с программированием.

Возможности анализа графа, отражающего интересы людей к другим людям, проектам с открытым исходным кодом в репозиториях и языкам программирования, огромны. Выбирая анализ для реализации, осмыслите природу задачи и извлекайте из графа только нужные данные — либо конкретное подмножество узлов с помощью метода `networkx.Graph.subgraph`, либо фильтруя узлы по типу или по некоторому пороговому значению.



Двусторонний анализ¹ пользователей и языков программирования заслуживает особого внимания, учитывая природу отношений между пользователями и языками. Двудольный граф включает два непересекающихся множества вершин и ребра, соединяющие два множества. На данном этапе легко можно удалить узлы репозитория из графа и тем самым существенно повысить эффективность вычисления глобальных статистик (число ребер уменьшится почти до 100 000).

¹ <http://bit.ly/1a100oP>.

8.4.5. Визуализация графа интересов

Визуализация графов — очень интересная тема, и часто картинка стоит тысячи слов, но имейте в виду, что не все графы с легкостью поддаются визуализации. Немного поразмыслив, нередко можно извлечь подграф, визуализация которого поможет получить более или менее полное представление о решаемой проблеме. Как вы уже знаете, граф — это всего лишь структура данных, и она не имеет какого-то определенного визуального представления. Для визуализации требуется использовать некоторый *алгоритм компоновки*, который отобразит узлы и ребра в двух- или трехмерное пространство в виде, пригодном для визуального восприятия.

Далее мы будем использовать все те же основные инструменты, что использовались на протяжении всей книги, и будем опираться на способность NetworkX экспортировать данные в формате JSON, поддающемся визуализации с применением JavaScript-библиотеки D3¹. Но имейте в виду, что существует множество других инструментов визуализации графов, достойных вашего внимания. К их числу относится Graphviz² — легко настраиваемый и широко известный инструмент, который способен преобразовывать очень сложные графы в растровые изображения. Он традиционно использовался в командной строке вместе с другими инструментами, но теперь имеет также графический интерфейс для большинства платформ. Другой вариант — Gephi³, еще один популярный проект с открытым исходным кодом, обладающий мощными интерактивными возможностями; за последние несколько лет популярность Gephi быстро выросла, и вам определенно стоит познакомиться с этим инструментом.

Пример 8.16 иллюстрирует шаблон извлечения подграфа с пользователями, отметившими начальный репозиторий, давший начало нашему графу (Mining-the-Social-Web), и связями «следует за» между ними. Затем он извлекает пользователей с общими интересами и визуализирует их ребра «следует за». Обратите внимание, что граф, сконструированный в этой главе, очень большой — он содержит десятки тысяч узлов и сотни тысяч ребер, поэтому постарайтесь получше разобраться в нем, чтобы добиться качественной визуализации с помощью такого инструмента, как Gephi.

¹ <http://bit.ly/1a1kGvo>.

² <http://bit.ly/1a1ooVG>.

³ <http://bit.ly/1a1opc5>.

Пример 8.16. Визуализация оригинального графа интересов социальной сети

```
import os
import json
from IPython.display import IFrame
from IPython.core.display import display
from networkx.readwrite import json_graph

print("Stats on the full graph")
print(nx.info(g))
print()

# Создать подграф из коллекции узлов. В данном случае подграф должен
# содержать всех пользователей из оригинального графа интересов

mtsw_users = [n for n in g if g.node[n]['type'] == 'user']
h = g.subgraph(mtsw_users)

print("Stats on the extracted subgraph")
print(nx.info(h))

# Визуализировать социальную сеть всех людей из оригинального графа интересов
d = json_graph.node_link_data(h)
json.dump(d, open('force.json', 'w'))

# Jupyter Notebook может обслуживать файлы и отображать их содержимое
# в плавающих фреймах. Добавьте в начало пути префикс 'files'.

# Шаблон D3 для отображения графов

viz_file = 'force.html'

# Отобразить полученное изображение

display(IFrame(viz_file, '100%', '500px'))
```

Получившийся результат показан на рис. 8.7.

8.5. Заключительные замечания

В предыдущих главах этой книги не раз упоминались разные виды графов, но только в этой главе было дано более подробное введение в их использование в роли гибкой структуры данных для представления сети пользователей GitHub и их общих интересов в отношении репозиторий программных проектов и языков программирования. Богатый программный интерфейс GitHub и простой

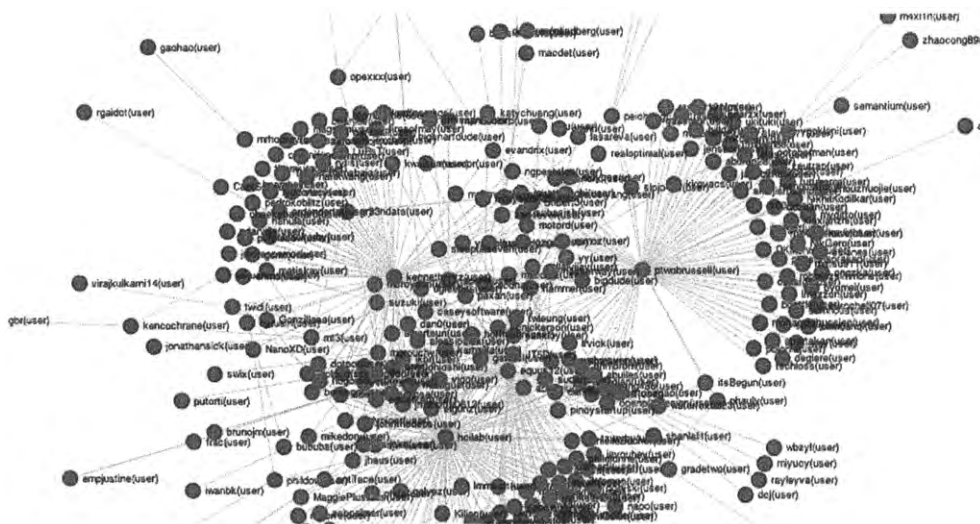


Рис. 8.7. Интерактивная карта ребер «следует за», связывающих пользователей GitHub в графе интересов: обратите внимание на закономерности в размещении узлов, которые соответствуют мерам центральности, описанным выше в этой главе

в использовании пакет NetworkX составили отличный дуэт для анализа интересных и часто упускаемых из виду социальных данных в одной из самых «социальных» сетей, широко распространившейся по всему миру. Понятие графа интересов не является новым, но его применение для организации социальной сети действительно является недавней разработкой с захватывающими последствиями. Если прежде графы интересов (или сопоставимые представления) использовались рекламодателями для эффективного размещения рекламы, то теперь они используются предпринимателями и разработчиками программного обеспечения для более эффективной ориентации на интересы и выработки рекомендаций, увеличивающих релевантность продуктов для пользователей.

Как и большинство других глав в книге, эта глава просто сыграла роль введения в графическое моделирование, графы интересов, GitHub API и возможные применения этих технологий. Методы графического моделирования, представленные в этой главе, с легкостью можно применить к другим социальным сетям, таким как Twitter или Facebook, и получить впечатляющие результаты, а также применить другие виды анализа к данным, доступным с использованием GitHub API. Возможности, как всегда, довольно обширны. Мы надеемся, что вы получили удовольствие от примеров в этой главе и узнали что-то новое, что сможете взять с собой в путешествие по миру анализа социальных сетей и не только.



Исходный код примеров для этой и всех других глав доступен на GitHub¹ в удобном формате Jupyter Notebook, который вы можете опробовать, не покидая веб-браузера.

8.6. Упражнения

- Повторите все примеры в этой главе, но в качестве отправной точки используйте другой репозиторий. Можете ли вы подтвердить верность выводов, сделанных в этой главе, или результаты ваших экспериментов имеют какие-то коренные отличия?
- В GitHub опубликовали некоторые данные о корреляции между языками программирования². Ознакомьтесь с этими данными. Отличаются ли они от того, что нам удалось выяснить с помощью GitHub API?
- Пакет NetworkX предлагает обширную коллекцию алгоритмов³ для обхода графов. Загляните в документацию и выберите пару алгоритмов для работы с данными. Хорошей отправной точкой могут послужить меры центральности, клики и двусторонние алгоритмы. Сможете ли вы найти в графе самую большую клику пользователей? А как насчет самой большой клики пользователей с таким общим интересом, как конкретный язык программирования?
- Архив GitHub Archive⁴ предлагает огромный объем данных о работе GitHub на глобальном уровне. Исследуйте эти данные с использованием некоторых инструментов, ориентированных на «большие данные».
- Проанализируйте два похожих проекта в GitHub и сравните результаты. Прекрасной отправной точкой для анализа могли бы стать Mining-the-Social-Web и Mining-the-Social-Web-2nd-Edition, учитывая неразрывную связь между ними. Найдите пользователей, которые отметили или создали ветвь из одного проекта, но не из другого. Как бы вы сравнили графы интересов? Попробуйте построить и проанализировать граф интересов, включающий всех пользователей, проявивших интерес к обоим изданиям.
- Используйте меру сходства, такую как мера сходства Жаккара (см. главу 4), для определения сходства двух произвольных пользователей GitHub

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² <http://bit.ly/1a1or3Y>.

³ <http://bit.ly/2GYXBDn>.

⁴ <http://bit.ly/1a1orAK>.

по таким признакам, как отмеченные репозитории, языки программирования и другие, которые вы сможете найти в GitHub API.

- Разработайте алгоритм, который вырабатывал бы рекомендации пользователям на основе информации о других пользователях и их интересах. В качестве основы можно было бы использовать код из раздела «Краткое введение в TF-IDF» главы 5, использующий косинусное сходство как меру релевантности прогноза.
- Постройте гистограмму, чтобы получить дополнительное представление о некотором аспекте графа интересов из этой главы, таком как популярность языков программирования.
- Познакомьтесь поближе с инструментами визуализации графов, такими как Graphviz и Gephi, позволяющими отображать содержимое графов в графическом представлении.
- Исследуйте набор данных «Friendster social network and ground-truth communities»¹ и проанализируйте его с применением алгоритмов NetworkX.

8.7. Онлайн-ресурсы

Ниже приводится список ссылок, упоминавшихся в этой главе, которые могут оказаться полезными для вас:

- Двудольный граф².
- Меры центральности³.
- Статья «Creating a personal access token for the command line»⁴.
- D3.js⁵.
- Набор данных «Friendster social network and ground-truth communities»⁶.
- Gephi⁷.

¹ <http://stanford.io/1a1orRr>.

² <http://bit.ly/1a1oooP>.

³ <http://bit.ly/1a1osEM>.

⁴ <http://bit.ly/1a1o7IG>.

⁵ <http://bit.ly/1a1kGvo>.

⁶ <http://stanford.io/1a1orRr>.

⁷ <http://bit.ly/1a1opc5>.

- GitHub Archive¹.
- GitHub Developer².
- Описание приемов постраничного извлечения результатов в документации для разработчиков GitHub³.
- Описание ограничений GitHub API в документации для разработчиков⁴.
- gitscm.com (онлайн-книга о Git)⁵.
- Видеолекция «Graph Theory — An Introduction!» на YouTube⁶.
- Graphviz⁷.
- Гиперграф⁸.
- Граф интересов⁹.
- Граф Krackhardt kite¹⁰.
- Алгоритм Краскала¹¹.
- Алгоритм вычисления минимального остовного дерева (Minimum Spanning Tree, MST)¹².
- NetworkX¹³.
- Алгоритмы обхода графов в NetworkX¹⁴.
- Репозиторий пакета PyGithub в GitHub¹⁵.

¹ <http://bit.ly/1a1orAK>.

² <http://bit.ly/1a1o49k>.

³ <http://bit.ly/1a1o9Ki>.

⁴ <http://bit.ly/1a1oblo>.

⁵ <http://bit.ly/1a1o2hZ>.

⁶ <http://bit.ly/1a1odto>.

⁷ <http://bit.ly/1a1ooVG>.

⁸ <http://bit.ly/1a1ocWm>.

⁹ <http://bit.ly/1a1o3Cu>.

¹⁰ <http://bit.ly/1a1oixa>.

¹¹ <http://bit.ly/1a1on3X>.

¹² <http://bit.ly/1a1omgr>.

¹³ <http://bit.ly/1a1ocFV>.

¹⁴ <http://bit.ly/2GYXBDn>.

¹⁵ <http://bit.ly/1a1o7Ca>.

Часть II

СБОРНИК РЕЦЕПТОВ ДЛЯ TWITTER

В отличие от первой части этой книги, где был дан общий обзор нескольких социальных сетей, в оставшейся главе мы вернемся к обсуждению Twitter и рассмотрим более двух десятков рецептов анализа данных из Twitter. Доступность программного интерфейса Twitter, присущая ему открытость и невероятная популярность делают эту социальную сеть идеальной для детального изучения, и в этой части книги мы создадим несколько простых строительных блоков, которые легко можно скомпоновать вместе для решения широкого спектра задач. Цель этого раздела — сосредоточить внимание на небольшом наборе решений типичных задач, которые можно адаптировать под особенности других социальных сетей. Так же как любой сборник технических рецептов, этот имеет удобную для навигации форму «задача/решение», и работая с ним, вы обязательно найдете интересные идеи, включая возможные настройки и модификации.

Мы настоятельно рекомендуем изучить предлагаемые рецепты, а когда вы придумаете свои необычные рецепты, поделитесь ими с сообществом книги, отправив запрос на включение в ее репозиторий GitHub, сообщите о нем в Twitter (и упомяните @SocialWebMining¹, если хотите, чтобы его ретвитнули) или на странице книги в Facebook².

¹ <http://bit.ly/1a1kHqz>.

² <http://on.fb.me/1a1kHPQ>.

9

Сборник рецептов для Twitter

Этот сборник содержит рецепты по анализу данных из Twitter. Каждый рецепт описывает решение одной конкретной задачи и максимально упрощен, чтобы дать возможность объединить несколько рецептов в одно целое с минимальными усилиями. Все рецепты можно рассматривать как элементарные строительные блоки, которые полезны сами по себе, но особенно полезны в сочетании с другими строительными блоками, которые сообща составляют более сложные единицы анализа. В отличие от предыдущих глав, содержащих больше теории, чем кода, здесь будет меньше теоретических рассуждений и больше программного кода. Идея главы заключается в том, чтобы дать вам возможность манипулировать фрагментами кода и объединять их для достижения своих собственных целей.

Большинство рецептов включают чуть больше кода, чем необходимо для выполнения параметризованных вызовов API и последующей обработки ответа, однако есть очень простые рецепты (состоящие всего из нескольких строк кода) и очень сложные. Этот сборник предназначен помочь вам, демонстрируя некоторые типичные задачи и их решения. В некоторых случаях может быть неочевидно, что необходимые данные можно получить, написав всего пару строк кода. Ценность предлагаемых вашему вниманию рецептов заключается в простоте их адаптации для ваших целей.

Одной из основных зависимостей программного обеспечения, которая вам понадобится во всех рецептах в этой главе, является пакет `twitter`, который можно установить с помощью диспетчера пакетов `pip`, выполнив очевидную команду **`pip install twitter`** в терминале. Другие зависимости будут отмечаться по мере их появления в отдельных рецептах. Если вы пользуетесь виртуальной машиной, созданной специально для этой книги (что мы настоятельно рекомендуем),

вам не придется ничего устанавливать, так как все необходимые зависимости уже установлены в ней.

Как вы узнали в главе 1, Twitter API v1.1 требует аутентификации всех запросов, поэтому предполагается, что вы последуете всем инструкциям в разделах «Доступ к Twitter API для целей разработки» и «Использование OAuth для доступа к Twitter API в промышленных целях», где предлагается сначала получить ключ аутентификации для подключения к API с последующим использованием его во всех рецептах.



Загрузите последний исправленный исходный код примеров для этой (и любой другой) главы, доступный по адресу: <http://bit.ly/Mining-the-Social-Web-3E>. Воспользуйтесь также преимуществами виртуальной машины, описанными в приложении А, чтобы получить максимальное удовольствие от опробования примеров кода.

9.1. Доступ к Twitter API для целей разработки

9.1.1. Задача

Вы собираетесь анализировать данные из своей учетной записи или решать иные задачи, и вам нужен простой способ доступа к API с целью разработки.

9.1.2. Решение

Для доступа к своей учетной записи посредством API без HTTP-переадресации используйте пакет `twitter` и учетные данные OAuth 2.0 из настроек приложения.

9.1.3. Пояснение

Twitter реализует OAuth 2.0, специально разработанный механизм авторизации, чтобы дать пользователям возможность предоставлять доступ к своим данным третьим сторонам, не совершая немемorable — передачи имени пользователя и пароля. Безусловно, для промышленных нужд можно использовать преимущества поддержки OAuth в Twitter, предлагая пользователям авторизовать

ваше приложение для доступа к их аккаунтам, но для нужд разработки можно пойти более простым путем и задействовать учетные данные из настроек вашего приложения, чтобы получить мгновенный доступ к своей собственной учетной записи.

Зарегистрируйте приложение в своей учетной записи Twitter, обратившись по адресу <http://dev.twitter.com/apps>, и запишите *ключ* и *секрет получателя*, а также *токен доступа* и *секрет токена доступа* — четыре элемента учетных данных, необходимых любому приложению с поддержкой OAuth 2.0 для доступа к учетной записи. На рис. 9.1 показан скриншот с настройками приложения в Twitter. Имея эти учетные данные, можно использовать любую библиотеку OAuth 2.0 для доступа к Twitter RESTful API¹, но в наших рецептах мы будем использовать пакет `twitter`, который реализует минималистичную обертку Python вокруг Twitter RESTful API. При регистрации приложения можно не указывать URL для обратной связи, потому что мы фактически идем в обход полноценной процедуры аутентификации OAuth и просто используем учетные данные для немедленного доступа к API. Пример 9.1 демонстрирует, как использовать эти учетные данные для создания соединения с API.

Пример 9.1. Доступ к Twitter API для целей разработки

```
import twitter

def oauth_login():
    # Перейдите по адресу http://twitter.com/apps/new, чтобы создать приложение
    # и получить учетные данные, которые необходимо подставить на место пустых
    # строк.
    # Дополнительную информацию о реализации OAuth в Twitter
    # ищите по адресу: https://dev.twitter.com/docs/auth/oauth.

    CONSUMER_KEY = ''
    CONSUMER_SECRET = ''
    OAUTH_TOKEN = ''
    OAUTH_TOKEN_SECRET = ''

    auth = twitter.oauth.OAuth(OAUTH_TOKEN, OAUTH_TOKEN_SECRET,
                               CONSUMER_KEY, CONSUMER_SECRET)

    twitter_api = twitter.Twitter(auth=auth)
    return twitter_api

# Пример использования
twitter_api = oauth_login()
```

¹ <http://bit.ly/1a1pDEq>.

```
# Попытка вывода переменной twitter_api ничего не даст, разве  
# только поможет увидеть, что переменная определена  
  
print(twitter_api)
```



Имейте в виду, что учетные данные, используемые для подключения, фактически равноценны комбинации «имя пользователя/пароль», поэтому всячески оберегайте их и в настройках приложения указывайте минимально необходимый уровень доступа. Для анализа данных вполне достаточно доступа только для чтения.

Этот способ удобно использовать для доступа к своим данным в своей учетной записи, но он не дает никаких преимуществ, если ваша цель — написать клиентское приложение для доступа к чужим данным. В этой ситуации вам придется выполнить полную процедуру аутентификации с использованием OAuth, как показано в примере 9.2.

9.2. Использование OAuth для доступа к Twitter API в промышленных целях

9.2.1. Задача

Требуется использовать OAuth, так чтобы приложение могло получить доступ к данным другого пользователя.

9.2.2. Решение

Реализуйте полный цикл авторизации с OAuth, используя пакет `twitter`.

9.2.3. Пояснение

Пакет `twitter` реализует полную процедуру аутентификации через OAuth для консольных приложений. Это достигается благодаря реализации метода *out of band* (oob), посредством которого приложение, выполняющееся вне браузера, например программа на Python, может безопасно получить четыре элемента учетных данных для доступа к API и запросить доступ к учетной записи конкретного пользователя. Если вы планируете написать веб-приложение,

обращающееся к учетной записи другого пользователя, может понадобиться немного адаптировать его реализацию.

Для реализации полного цикла аутентификации через OAuth из Jupyter Notebook не так много причин (если только вы не планируете задействовать службу Jupyter Notebook для использования другими людьми), поэтому в данном рецепте используется фреймворк Flask, играющий роль встроенного веб-сервера, для демонстрации процесса и тот же набор инструментов, что применяется в остальной книге. Его легко адаптировать для работы с любым фреймворком веб-приложений, поскольку основная идея остается неизменной.

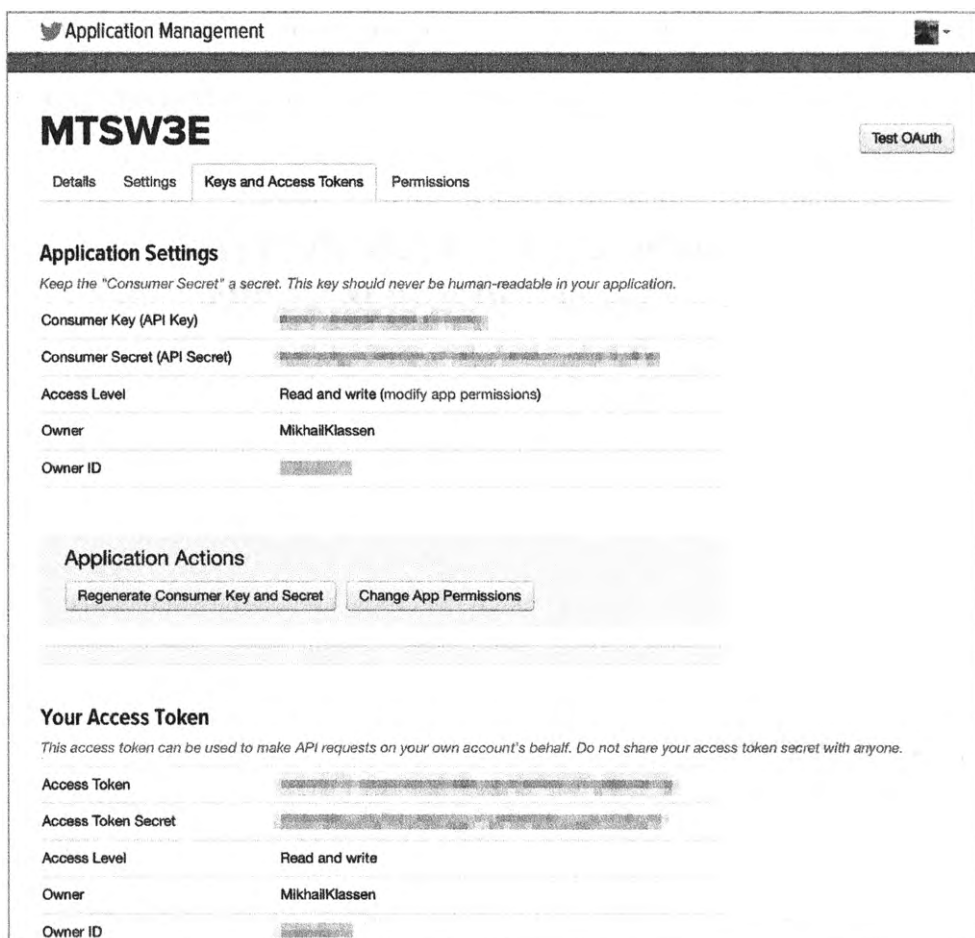


Рис. 9.1. Пример настроек OAuth для приложения Twitter

На рис. 9.1 показан скриншот с настройками приложения Twitter. Значения **Consumer key** (Ключ получателя) и **Consumer secret** (Секрет получателя), описанные выше в разделе «Доступ к Twitter API для целей разработки», однозначно идентифицируют приложение при использовании протокола OAuth 2.0. Вы должны передать эти значения с запросом на доступ к данным пользователя, чтобы Twitter мог запросить у пользователя одобрение и передать ему информацию о характере вашего запроса. Предположим, что пользователь одобрил запрос приложения. В этом случае Twitter обратится к URL обратного вызова, указанному в настройках приложения, и включит *верификатор OAuth* для получения *токена доступа* и *секрета токена доступа, используемых совместно с ключом и секретом потребителя*, чтобы в конечном счете дать вашему приложению доступ к данным учетной записи. (При использовании метода аутентификации oob OAuth не требуется указывать URL обратного вызова; Twitter предоставит пользователю PIN-код для верификации в OAuth, который необходимо вручную скопировать в приложение.) Дополнительные сведения о процедуре аутентификации с использованием OAuth 2.0 вы найдете в приложении Б.

Пример 9.2 иллюстрирует, как использовать ключ и секрет получателя для аутентификации посредством OAuth и с использованием пакета `twitter` и получить доступ к пользовательским данным. Токен доступа и секрет токена доступа записываются на диск, что упрощает авторизацию в будущем. Как описывается в сборнике вопросов¹, часто задаваемых разработчиками Twitter, в настоящее время срок действия токенов не ограничивается, а значит, их можно хранить и использовать от имени пользователя неограниченное количество времени при соблюдении условий обслуживания².

Пример 9.2. Процесс аутентификации посредством OAuth для доступа к Twitter API в промышленных целях

```
import json
from flask import Flask, request
import multiprocessing
from threading import Timer
from IPython.display import IFrame
from IPython.display import display
from IPython.display import Javascript as JS

import twitter
from twitter.oauth_dance import parse_oauth_tokens
```

¹ <http://bit.ly/2Lzux3x>.

² <http://twitter.com/ru/tos>.

```
from twitter.oauth import read_token_file, write_token_file

# Примечание: Этот код в точности повторяет процедуру, реализованную
# в блокноте _AppendixB

OAUTH_FILE = "resources/ch09-twittercookbook/twitter_oauth"

# Перейдите по адресу http://twitter.com/apps/new, чтобы создать приложение
# и получить учетные данные, которые необходимо подставить на место пустых
# строк.
# Дополнительную информацию о реализации OAuth в Twitter ищите по адресу:
# https://developer.twitter.com/en/docs/basics/authentication/overview/oauth
# Если вы используете фреймворк Flask с Jupyter Notebook,
# определите URL обратного вызова в настройках вашего приложения,
# присвоенный переменной *oauth_callback* ниже.

# Определение нескольких переменных, которые будут перетекать в лексическую
# область видимости пары функций, следующих ниже
CONSUMER_KEY = ''
CONSUMER_SECRET = ''
oauth_callback = 'http://127.0.0.1:5000/oauth_helper'

# Настройка обработчика обратного вызова, который выполнит Twitter после того,
# как пользователь авторизует приложение

webserver = Flask("TwitterOAuth")
@webserver.route("/oauth_helper")
def oauth_helper():

    oauth_verifier = request.args.get('oauth_verifier')

    # Сохранить учетные данные из ipynb_oauth_dance
    oauth_token, oauth_token_secret = read_token_file(OAUTH_FILE)

    _twitter = twitter.Twitter(
        auth=twitter.OAuth(
            oauth_token, oauth_token_secret, CONSUMER_KEY, CONSUMER_SECRET),
        format='', api_version=None)

    oauth_token, oauth_token_secret = parse_oauth_tokens(
        _twitter.oauth.access_token(oauth_verifier=oauth_verifier))

    # Этот веб-сервер необходим только для обработки единственного запроса,
    # поэтому остановить его
    shutdown_after_request = request.environ.get('werkzeug.server.shutdown')
    shutdown_after_request()

    # Сохранить окончательные учетные данные, которые затем можно прочитать
    # после блокирующего вызова webserver.run()
```

```
write_token_file(OAUTH_FILE, oauth_token, oauth_token_secret)

return "%s %s written to %s" % (oauth_token, oauth_token_secret, OAUTH_FILE)

# Для работы с реализацией Twitter OAuth 1.0a достаточно реализовать свой
# цикл "аутентификации через oauth", в котором мы будем точно следовать
# шаблону, реализованному в twitter.oauth_dance

def ipynb_oauth_dance():
    _twitter = twitter.Twitter(
        auth=twitter.OAuth('', '', CONSUMER_KEY, CONSUMER_SECRET),
        format='', api_version=None)

    oauth_token, oauth_token_secret = parse_oauth_tokens(
        _twitter.oauth.request_token(oauth_callback=oauth_callback))

    # Эти промежуточные значения нужно записать в файл, чтобы прочитать их
    # при обработке обратного вызова из Twitter веб-сервером в /oauth_helper
    write_token_file(OAUTH_FILE, oauth_token, oauth_token_secret)

    oauth_url = ('http://api.twitter.com/oauth/authorize?oauth_token=' +
                 oauth_token)

    # Использовать встроенные средства веб-браузера для доступа к веб-серверу через
    # новое окно, чтобы авторизовать пользователя
    display(JS("window.open('%s')" % oauth_url))

    # После завершения блокирующего вызова webserver.run() начать процедуру OAuth,
    # которая в конечном итоге заставит Twitter переадресовать запрос обратно.
    # После обработки этого запроса веб-сервер завершится и программа продолжит
    # выполнение с файлом OAUTH_FILE, содержащим необходимые учетные данные.
    Timer(1, lambda: ipynb_oauth_dance()).start()

webserver.run(host='0.0.0.0')

# Значения, прочитанные из этого файла, были добавлены
# в конец /oauth_helper
oauth_token, oauth_token_secret = read_token_file(OAUTH_FILE)

# Эти четыре значения необходимы для авторизации приложения
auth = twitter.oauth.OAuth(oauth_token, oauth_token_secret,
                           CONSUMER_KEY, CONSUMER_SECRET)

twitter_api = twitter.Twitter(auth=auth)

print(twitter_api)
```

Обратите внимание, что токен доступа и секрет токена доступа, которые получает ваше приложение, совпадают со значениями в настройках приложения,

и это не случайно. Оберегайте эти значения, так как владение ими равносильно владению именем пользователя и пароля.

9.3. Поиск актуальных тем

9.3.1. Задача

Требуется определить актуальные темы, обсуждаемые в определенном географическом регионе, например в Соединенных Штатах, в другой стране, в группе стран или даже во всем мире.

9.3.2. Решение

Программный интерфейс Twitter Trends API¹ позволяет определить актуальные темы, обсуждаемые в конкретной географической области, определяемой идентификатором, который можно получить, обратившись к службе Where On Earth (WOE)², первоначально созданной компанией Yahoo! и называвшейся GeoPlanet.

9.3.3. Пояснение

Местоположение — важнейшее понятие для платформы разработки Twitter. Соответственно, актуальные темы ограничены географическими регионами, чтобы обеспечить максимально удобный программный интерфейс для запросов на получение актуальных тем (как показано в примере 9.3). Подобно всем другим API, он возвращает список актуальных тем в виде данных JSON, которые можно преобразовать в стандартные объекты Python и затем манипулировать ими с использованием генераторов списков и других похожих инструментов. Это значительно упрощает анализ ответов API. Поэкспериментируйте с разными идентификаторами WOE и сравните темы, считающиеся актуальными в разных географических регионах. Например, сравните темы, обсуждаемые в двух разных странах, или темы, обсуждаемые в одной стране и во всем мире.

¹ <http://bit.ly/2jSxPmY>.

² <http://bit.ly/2jVlcXo>.

Пример 9.3. Поиск актуальных тем

```
import json
import twitter

def twitter_trends(twitter_api, woe_id):
    # Обратите внимание на символ подчеркивания перед именем параметра id
    # со строкой, параметризующей запрос. Если передать его без символа
    # подчеркивания, пакет twitter добавит значение параметра в сам URL
    # как специальный именованный аргумент.
    return twitter_api.trends.place(_id=woe_id)

# Пример использования

twitter_api = oauth_login()

# Другие идентификаторы Where On Earth вы найдете по адресам:
# https://bit.ly/2pdi0tS и http://www.woeidlookup.com

WORLD_WOE_ID = 1
world_trends = twitter_trends(twitter_api, WORLD_WOE_ID)
print(json.dumps(world_trends, indent=1))

US_WOE_ID = 23424977
us_trends = twitter_trends(twitter_api, US_WOE_ID)
print(json.dumps(us_trends, indent=1))
```

9.4. Поиск твитов

9.4.1. Задача

Требуется найти твиты, используя ключевые слова и определенные ограничения.

9.4.2. Решение

Используйте Search API.

9.4.3. Пояснение

Для поиска твитов во вселенной Twitter можно использовать Search API¹. Подобно многим другим поисковым системам, Twitter Search API возвращает

¹ <http://bit.ly/2IcgdRL>.

результаты пакетами, при этом есть возможность настроить максимальное число результатов в пакете, передав число от 1 до 200 в именованном параметре `count`. Весьма вероятно, что в ответ на запрос будет найдено результатов больше 200 (или числа, указанного в параметре `count`), и в этом случае, выражаясь терминологией Twitter API, для получения следующего пакета вам придется использовать *курсор*.

*Курсоры*¹ — это новое расширение, появившееся в Twitter API v1.1. Они обеспечивают более надежную схему, чем парадигма страничного извлечения данных, реализованная в API v1.0, которая требует указывать номер страницы и число результатов на странице. Парадигма курсоров лучше приспособлена к динамической природе платформы Twitter. Например, курсоры Twitter API изначально учитывают возможность появления новой информации в режиме реального времени во время навигации по пакетам с результатами поиска. Иначе говоря, может случиться так, что в процессе навигации по пакетам с результатами может появиться новая информация, соответствующая запросу, которую вы хотели бы включить в текущие результаты в ходе навигации вместо отправки нового запроса.

Пример 9.4 демонстрирует, как использовать Search API и курсор, чтобы получить несколько пакетов с результатами.

Пример 9.4. Поиск твитов

```
def twitter_search(twitter_api, q, max_results=200, **kw):
    # Дополнительную информацию о дополнительных критериях, которые
    # можно передавать в именованных аргументах, ищите по адресам:
    # http://bit.ly/2QyGz0P и https://bit.ly/2QyGz0P

    # См. https://dev.twitter.com/docs/api/1.1/get/search/tweets
    search_results = twitter_api.search.tweets(q=q, count=100, **kw)

    statuses = search_results['statuses']

    # Извлекать пакеты с результатами с использованием курсора, пока не будет
    # получено желаемое число результатов. Не забывайте, что пользователи,
    # авторизованные через OAuth, могут выполнить "только" 180 запросов в
    # течение 15-минутного интервала. Подробности ищите по адресу:
    # https://developer.twitter.com/en/docs/basics/rate-limits.
    # Число результатов ~1000 выглядит вполне разумным, хотя не для всех
    # запросов будет найдено такое число результатов.

    # Установите разумное ограничение
```

¹ <http://bit.ly/2IEOvf1>.

```
max_results = min(1000, max_results)

for _ in range(10): # 10*100 = 1000
    try:
        next_results = search_results['search_metadata']['next_results']
    except KeyError as e: # No more results when next_results doesn't exist
        break

    # Создать словарь из строки next_results, имеющей следующую форму:
    # ?max_id=313519052523986943&q=NCAA&include_entities=1
    kwargs = dict([ kv.split('=')
                    for kv in next_results[1:].split("&") ])

    search_results = twitter_api.search.tweets(**kwargs)
    statuses += search_results['statuses']

    if len(statuses) > max_results:
        break

return statuses

# Пример использования
twitter_api = oauth_login()

q = "CrossFit"
results = twitter_search(twitter_api, q, max_results=10)

# Вывести один пример результата, получив срез списка...
print(json.dumps(results[0], indent=1))
```

9.5. Конструирование удобных вызовов функций

9.5.1. Задача

Требуется связать определенные параметры с вызовами функций и передавать ссылки на связанные функции для упрощения программного кода.

9.5.2. Решение

Используйте `functools.partial` для создания полностью или частично связанных функций, которые можно передавать другому коду и вызывать их там без передачи дополнительных параметров.

9.5.3. Пояснение

Прием, основанный на применении `functools.partial`, не имеет прямого отношения к шаблонам программирования Twitter API, но его очень удобно использовать в комбинации с пакетом `twitter`, а также во многих других шаблонах программирования в этом сборнике рецептов и на языке Python в целом. Например, порой слишком утомительно снова и снова передавать ссылку на соединение с Twitter API (переменная `twitter_api` в этих рецептах, которая передается большинству функций в первом аргументе), и было бы желательно иметь функцию, определяющую часть аргументов, чтобы можно было передавать и вызывать ее, указывая только недостающие параметры. Точно так же, если вы устали раз за разом вводить `json.dumps(..., indent=1)`, вы могли бы определить именованный аргумент один раз и дать функции более короткое имя, например `pp` (`pretty-print` — форматированный вывод), чтобы избавить себя от дополнительного ввода с клавиатуры.

Другой пример, иллюстрирующий удобство частично связанных параметров: можно связать соединение с Twitter API и идентификатор WOE географического региона в Trends API с единственной функцией, чтобы затем просто передавать и вызывать ее без параметров. Возможности в этом плане огромны, и несмотря на то что тот же эффект можно получить, определяя функции с использованием ключевого слова `def`, способ на основе `functools.partial` в некоторых ситуациях выглядит лаконичнее и элегантнее. Пример 9.5 демонстрирует несколько вариантов, которые могут вам пригодиться.

Пример 9.5. Конструирование удобных вызовов функций

```
from functools import partial

pp = partial(json.dumps, indent=1)

twitter_world_trends = partial(twitter_trends, twitter_api, WORLD_WOE_ID)
print(pp(twitter_world_trends()))

authenticated_twitter_search = partial(twitter_search, twitter_api)
results = authenticated_twitter_search("iPhone")
print(pp(results))

authenticated_iphone_twitter_search = partial(authenticated_twitter_search,
"iPhone")
results = authenticated_iphone_twitter_search()
print(pp(results))
```

9.6. Запись и чтение текстовых файлов с данными JSON

9.6.1. Задача

Следует сохранить относительно небольшой объем данных, полученных из Twitter API, для последующего анализа или архивирования.

9.6.2. Решение

Запишите данные в текстовый файл в удобном и переносимом формате JSON.

9.6.3. Пояснение

Текстовые файлы подходят не для всех случаев, тем не менее они удобны, когда требуется просто сбросить данные на диск, чтобы сохранить их для последующих экспериментов или анализа. На самом деле этот прием является оптимальным решением, так как позволяет минимизировать число запросов к Twitter API и избежать проблем с ограничениями, с которыми вы можете столкнуться. В конце концов, не в ваших интересах и не в интересах Twitter снова и снова посылать запросы к API, чтобы получить те же самые данные.

Пример 9.6 демонстрирует довольно типичное использование пакета `io` для сохранения данных на диск и чтения их с диска в кодировке UTF-8, чтобы избежать (часто удручающего и не всегда понятного) исключения `UnicodeDecodeError`, обычно возникающего при сериализации и десериализации данных в приложениях на Python.

Пример 9.6. Запись и чтение текстовых файлов с данными JSON

```
import io, json

def save_json(filename, data):
    with open('resources/ch09-twittercookbook/{0}.json'.format(filename),
              'w', encoding='utf-8') as f:
        json.dump(data, f, ensure_ascii=False)

def load_json(filename):
```

```
with open('resources/ch09-twittercookbook/{0}.json'.format(filename),
          'r', encoding='utf-8') as f:
    return json.load(f)

# Пример использования
q = 'CrossFit'

twitter_api = oauth_login()
results = twitter_search(twitter_api, q, max_results=10)

save_json(q, results)
results = load_json(q)

print(json.dumps(results, indent=1, ensure_ascii=False))
```

9.7. Сохранение данных JSON в MongoDB и доступ к ним

9.7.1. Задача

Требуется сохранить нетривиальный объем данных JSON, полученных из Twitter API, и организовать доступ к ним.

9.7.2. Решение

Для сохранения данных в удобном формате JSON используйте документоориентированную базу данных, такую как MongoDB.

9.7.3. Пояснение

Для хранения небольших объемов данных в файлах JSON вполне можно использовать каталог в файловой системе, однако вас может удивить, с какой скоростью могут накапливаться данные и быстро превысить порог, за которым работа с простыми текстовыми файлами превращается в утомительное занятие. К счастью, существуют документоориентированные базы данных, такие как MongoDB, которые идеально подходят для хранения ответов Twitter API, потому что они предназначены для эффективного хранения данных JSON.

MongoDB — надежная и хорошая документированная база данных, прекрасно справляющаяся и с небольшими, и с огромными объемами данных. Она поддерживает мощные операторы запросов и имеет механизмы индексирования, которые существенно упрощают реализацию анализа в коде на Python.



MongoDB можно установить на большинство платформ¹ и для этой базы данных имеется превосходная онлайн-документация², описывающая процедуры установки/настройки и выполнения запросов/индексирования.

В большинстве случаев, когда требуется индексировать и запрашивать данные, производительность MongoDB может оказаться намного выше производительности вашего кода благодаря ее механизму индексирования и использованию эффективного формата JSON представления данных на диске. Пример 9.7 иллюстрирует, как подключиться к действующей базе данных MongoDB, чтобы сохранить и извлечь данные.



Второе издание этой книги содержало довольно обширное введение в MongoDB в главе 7, в контексте сохранения (содержимого почтового ящика в формате JSON) данных и использования механизма агрегирования³ в MongoDB для выполнения нетривиальных запросов. Оно было убрано из текущего издания, чтобы больше места выделить для обсуждения библиотеки `pandas`, которая, по мнению авторов, является важнейшим инструментом в арсенале любого исследователя данных. Любую задачу можно решить несколькими способами, и у каждого из нас есть свои любимые инструменты.

Пример 9.7. Сохранение данных JSON в MongoDB и доступ к ним

```
import json
import pymongo # pip install pymongo

def save_to_mongo(data, mongo_db, mongo_db_coll, **mongo_conn_kw):

    # Подключиться к серверу MongoDB, по умолчанию доступному по адресу
    # localhost:27017

    client = pymongo.MongoClient(**mongo_conn_kw)

    # Получить ссылку на указанную базу данных
```

¹ <http://bit.ly/2jUeG3Z>.

² <http://bit.ly/2Ih7bmn>.

³ <http://bit.ly/1a1pGjv>.

```
db = client[mongo_db]

# Получить ссылку на указанную коллекцию в базе данных
coll = db[mongo_db_coll]

# Записать большой объем данных и вернуть идентификаторы
try:
    return coll.insert_many(data)
except:
    return coll.insert_one(data)

def load_from_mongo(mongo_db, mongo_db_coll, return_cursor=False,
                    criteria=None, projection=None, **mongo_conn_kw):

    # Для ограничения возвращаемых данных можно использовать критерии
    # и проекции, как описывается в документации:
    # http://docs.mongodb.org/manual/reference/method/db.collection.find/

    # Для выполнения более сложных запросов можно использовать механизм
    # агрегирования MongoDB

    client = pymongo.MongoClient(**mongo_conn_kw)
    db = client[mongo_db]
    coll = db[mongo_db_coll]

    if criteria is None:
        criteria = {}

    if projection is None:
        cursor = coll.find(criteria)
    else:
        cursor = coll.find(criteria, projection)

    # Большие объемы данных предпочтительнее возвращать в виде курсора

    if return_cursor:
        return cursor
    else:
        return [ item for item in cursor ]

# Пример использования

q = 'CrossFit'

twitter_api = oauth_login()
results = twitter_search(twitter_api, q, max_results=10)

ids = save_to_mongo(results, 'search_results', q)

load_from_mongo('search_results', q)
```

При желании данные из Twitter можно сохранить в формате `DataFrame`, поддерживаемом библиотекой `pandas`. Этот способ демонстрирует пример 9.8. Этот подход с успехом можно использовать в проектах, выполняющих анализ малых и средних объемов данных, но если объем данных превышает емкость оперативной памяти компьютера (ОЗУ), придется найти какое-то другое решение. А как только объем данных превысит емкость жесткого диска, вам понадобится решение на основе распределенной базы данных. Это уже сфера больших данных.

Пример 9.8. Сохранение данных JSON и доступ к ним с помощью `pandas`

```
import json
import pickle
import pandas as pd

def save_to_pandas(data, fname):
    df = pd.DataFrame.from_records(data)
    df.to_pickle(fname)
    return df

def load_from_mongo(fname):
    df = pd.read_pickle(fname)
    return df

# Пример использования

q = 'CrossFit'

twitter_api = oauth_login()
results = twitter_search(twitter_api, q, max_results=10)

df = save_to_pandas(results, 'search_results_{}.pkl'.format(q))

df = load_from_mongo('search_results_{}.pkl'.format(q))

# Вывести для примера часть данных, только столбцы user и text
df[['user', 'text']].head()
```

9.8. Получение выборки из потока твитов с использованием Streaming API

9.8.1. Задача

Требуется проанализировать, о чем люди пишут *прямо сейчас*, выбирая твиты из потока в масштабе реального времени, не обращаясь с запросом к Search API,

который может возвращать немного (или сильно) устаревшую информацию. Или требуется начать накопление нетривиального объема данных по конкретной теме для последующего анализа.

9.8.2. Решение

Используйте Twitter Streaming API¹ для выборки общедоступных данных из общего потока.

9.8.3. Пояснение

Twitter случайным образом выбирает до 1% от всех твитов, поступающих в режиме реального времени, и предоставляет к ним доступ через потоковый программный интерфейс Streaming API. Если только вы не предполагаете получить доступ к Twitter Enterprise API² или воспользоваться услугами стороннего провайдера, такого как DataSift³ (которые во многих случаях стоят очень дорого), это решение может оказаться лучшим. На первый взгляд может показаться, что 1% — это ничтожно мало, но, потратив немного времени, вы заметите, что в пиковые моменты скорость поступления твитов может достигать десятков тысяч в секунду. В случае с обширными темами сохранение всех твитов может быстро превратиться в более сложную проблему, чем вы думаете. 1% от всех твитов — это очень много.

В отличие от Search API, который немного проще в использовании и возвращает «историческую» информацию (во вселенной Twitter это может означать, что она опубликована минуты или часы тому назад, в зависимости от того, как быстро темы набирают и теряют актуальность), программный интерфейс Streaming API предлагает возможность получения выборки из *глобального потока информации* в режиме, очень близком к реальному времени. Пакет `twitter` предлагает простую и удобную поддержку Streaming API, позволяющую фильтровать поток твитов по ключевым словам и дающую понятный и удобный способ доступа к информации. В этом случае вместо объекта соединения `twitter.Twitter` нужно создать объект `twitter.TwitterStream`, передав конструктору именованный аргумент со ссылкой на экземпляр типа `twitter.oauth.OAuth`, обсуждавшегося в разделах «Доступ к Twitter API для целей разработки» и «Использование OAuth для доступа к Twitter API в промышленных целях».

¹ <http://bit.ly/2rDU17W>.

² <http://bit.ly/2KZ1mrJ>.

³ <http://bit.ly/1a1pGQE>.

Код в примере 9.9 демонстрирует, как начать использовать Twitter Streaming API.

Пример 9.9. Получение выборки из потока твитов с использованием Streaming API

Отыскивает нужные темы с использованием средств фильтрации, предлагаемых API

```
import sys
import twitter
```

Критерии запроса

```
q = 'CrossFit' # Список критериев через запятую
```

```
print('Filtering the public timeline for track={0}'.format(q), file=sys.stderr)
sys.stderr.flush()
```

```
# Получить экземпляр twitter.Twitter
twitter_api = oauth_login()
```

```
# Получить ссылку на параметр self.auth
twitter_stream = twitter.TwitterStream(auth=twitter_api.auth)
```

```
# См. https://developer.twitter.com/en/docs/tutorials/consuming-streaming-data
stream = twitter_stream.statuses.filter(track=q)
```

```
# Например, если не выходит получить хоть что-то, попробуйте
# искать по имени Justin Bieber, и вы обязательно
# что-нибудь да получите (по крайней мере в Twitter)
```

```
for tweet in stream:
    print(tweet['text'])
    sys.stdout.flush()
```

```
# Сохранить в базе данных в определенной коллекции
```

9.9. Сбор временных последовательностей данных

9.9.1. Задача

Требуется периодически посылать запросы Twitter API для получения конкретных результатов или актуальных тем и сохранять данные для последующего анализа временных рядов.

9.9.2. Решение

Используйте встроенную функцию `time.sleep` внутри бесконечного цикла для выполнения запросов и сохранения результатов в базе данных, такой как MongoDB, если прием использования Streaming API, как описано в разделе «Получение выборки из потока твитов с использованием Streaming API», по какой-то причине не подходит.

9.9.3. Пояснение

Кроме простой возможности выполнять точечные запросы по конкретным ключевым словам в конкретные моменты времени, существует также возможность накапливать данные с течением времени и определять тренды и закономерности, открывающая доступ к очень мощной разновидности анализа, которую часто упускают из виду. Каждый раз, когда вы оглядываетесь назад и говорите: «Если бы я знал...», — это можно расценивать как признак ситуации, когда можно было бы заранее собрать данные и использовать для экстраполяции или прогнозирования будущего (там, где это возможно).

Анализ временных рядов применительно к данным из Twitter может быть настоящим захватывающим, учитывая появление и угасание актуальных тем. Часто бывает полезно получить выборку из потока твитов и сохранить результаты в документоориентированной базе данных, такой как MongoDB, но иногда проще или целесообразнее выполнять запросы и записывать результаты через определенные интервалы времени. Например, можно запрашивать актуальные темы для разных географических регионов в течение суток и измерять частоту изменения трендов, сравнивать скорость изменений в разных регионах, отыскивать самые длительные и самые короткие тренды и многое другое.

Другая интересная возможность, которая активно исследуется, — выявление корреляции между настроениями, выражаемыми в Twitter, и состоянием фондовых рынков. Достаточно легко подобрать определенные ключевые слова, хештеги или актуальные темы и затем сопоставить данные с фактическими изменениями фондового рынка; это может стать шагом к созданию бота, который будет делать прогнозы относительно фондовых и товарно-сырьевых рынков.

Код в примере 9.10 фактически объединяет код из примера в разделе «Доступ к Twitter API для целей разработки» с кодом из примеров 9.3 и 9.7 и демонстрирует, как можно использовать эти рецепты в качестве простейших строитель-

ных блоков для создания более сложных сценариев, проявив совсем немного творческой смекалки.

Пример 9.10. Сбор временных последовательностей данных

```
import sys
import datetime
import time
import twitter

def get_time_series_data(api_func, mongo_db_name, mongo_db_coll,
                        secs_per_interval=60, max_intervals=15, **mongo_conn_kw):

    # Настройки по умолчанию, 15 интервалов и 1 вызов API в каждом интервале,
    # гарантируют, что вы не превысите ограничение, накладываемое Twitter.

    interval = 0

    while True:

        # Текущее время в формате "2013-06-14 12:52:07"
        now = str(datetime.datetime.now()).split(".")[0]

        response = save_to_mongo(api_func(), mongo_db_name,
                                mongo_db_coll + "-" + now, **mongo_conn_kw)

        print("Write {0} trends".format(len(response.inserted_ids)),
              file=sys.stderr)

        print("Zzz...", file=sys.stderr)
        sys.stderr.flush()

        time.sleep(secs_per_interval) # seconds
        interval += 1

        if interval >= 15:
            break

# Пример использования
get_time_series_data(twitter.world_trends, 'time-series', 'twitter_world_trends')
```

9.10. Извлечение сущностей из твитов

9.10.1. Задача

Требуется извлечь из твитов сущности, такие как @username, хештеги и URL, для дальнейшего анализа.

9.10.2. Решение

Извлеките сущности из поля `entities` твитов.

9.10.3. Пояснение

Twitter API возвращает сущности, упоминаемые в твитах, в стандартном поле, содержащемся в большинстве ответов, где это применимо. Поле `entities`, используемое в примере 9.11, включает упоминания пользователей, хештеги, адреса URL, медиаобъекты (такие, как изображения и видео) и финансовые символы, такие как обозначения ценных бумаг. В настоящее время в некоторых ситуациях какие-то поля могут отсутствовать. Например, поле `media` присутствует и заполнено, только если пользователь вставил медиаобъект с помощью клиента Twitter, который использует для этого API, специально предназначенный для встраивания медиаконтента; простая вставка ссылки на видео в YouTube не влечет заполнение этого поля.

За дополнительной информацией, в том числе о дополнительных полях для сущностей разных типов, обращайтесь к документации с описанием API¹. Например, для адресов URL Twitter предлагает несколько вариантов, включая сокращенные и полные формы, а также значения, которые иногда могут лучше подходить для отображения в пользовательском интерфейсе.

Пример 9.11. Извлечение сущностей из твитов

```
def extract_tweet_entities(statuses):  
    # Дополнительную информацию о твитах ищите в документации по адресу:  
    # https://bit.ly/2MELMkm  
  
    if len(statuses) == 0:  
        return [], [], [], [], []  
  
    screen_names = [ user_mention['screen_name']  
                     for status in statuses  
                     for user_mention in status['entities']['user_mentions']  
    ]  
  
    hashtags = [ hashtag['text']  
                 for status in statuses  
                 for hashtag in status['entities']['hashtags'] ]  
  
    urls = [ url['expanded_url']
```

¹ <http://bit.ly/2wD3VfB>.

```
        for status in statuses
            for url in status['entities']['urls'] ]

# В некоторых случаях (например, в результатах поиска) медиасущности
# могут отсутствовать
medias = []
symbols = []
for status in statuses:
    if 'media' in status['entities']:
        for media in status['entities']['media']:
            medias.append(media['url'])
    if 'symbol' in status['entities']:
        for symbol in status['entities']['symbol']:
            symbols.append(symbol)

return screen_names, hashtags, urls, medias, symbols

# Пример использования

q = 'CrossFit'

statuses = twitter_search(twitter_api, q)

screen_names, hashtags, urls, media, symbols = extract_tweet_entities(statuses)

# Исследовать первые пять элементов...

print(json.dumps(screen_names[0:5], indent=1))
print(json.dumps(hashtags[0:5], indent=1))
print(json.dumps(urls[0:5], indent=1))
print(json.dumps(media[0:5], indent=1))
print(json.dumps(symbols[0:5], indent=1))
```

9.11. Поиск самых популярных твитов в коллекции

9.11.1. Задача

Требуется определить самые популярные твиты в коллекции, например, в результатах поиска или в пользовательской ленте сообщений.

9.11.2. Решение

Проверьте поле `retweet_count` твитов, чтобы определить, был ли тот или иной твит ретвитнут и количество ретвитов.

9.11.3. Пояснение

Анализ поля `retweet_count` твита, как показано в примере 9.12, является, пожалуй, самым простым способом оценки популярности, потому что пользователи обязательно будут делиться с другими популярными твитами. В зависимости от значения понятия «популярный» в формулу определения популярности твита можно также добавить поле `favorite_count`, определяющее, сколько раз твит был добавлен пользователями в закладки. Например, значение в поле `retweet_count` можно взвесить коэффициентом 1,0, а значение в поле `favorite_count` — коэффициентом 0,1, чтобы добавить веса твитам, которые были ретвитнуты и добавлены в закладки.

Выбор конкретных коэффициентов в формуле полностью возлагается на вас и будет зависеть от того, насколько важную роль, по вашему мнению, играет каждое поле в решении поставленной задачи. Также в некоторых случаях может быть полезно реализовать экспоненциальное затухание, когда более свежие твиты получают больший вес.



См. также примеры 9.14 и 9.15, где описываются дополнительные детали, касающиеся навигации по пространству анализа и определения первоначального автора ретвита, которые могут быть более запутанными, чем кажется на первый взгляд.

Пример 9.12. Поиск самых популярных твитов в коллекции

```
import twitter

def find_popular_tweets(twitter_api, statuses, retweet_threshold=3):

    # В вычисление этой эвристики также можно добавить поле favorite_count,
    # чтобы добавить веса популярным твитам при ранжировании

    return [ status
            for status in statuses
              if status['retweet_count'] > retweet_threshold ]

# Пример использования

q = "CrossFit"

twitter_api = oauth_login()
search_results = twitter_search(twitter_api, q, max_results=200)
```

```
popular_tweets = find_popular_tweets(twitter_api, search_results)
for tweet in popular_tweets:
    print(tweet['text'], tweet['retweet_count'])
```



Атрибут `retweeted` твита *не* является быстрым способом определить, был ли ретвитнут данный твит. Это поле, которое называют «зависящим от точки зрения» (*perspectival*), сообщает, был ли твит ретвитнут текущим аутентифицированным пользователем (которым являетесь вы сами, если анализируете собственные данные), что может быть удобно для отображения маркеров в пользовательском интерфейсе. Его называют зависящим от точки зрения, потому что его значение определяется с точки зрения аутентифицированного пользователя.

9.12. Поиск самых популярных сущностей в коллекции твитов

9.12.1. Задача

Требуется определить самые популярные сущности, такие как `@username`, `#хештеги` или адреса URL в коллекции твитов, что может дать дополнительное понимание природы этой коллекции.

9.12.2. Решение

Извлеките сущности с помощью генератора списков, подсчитайте и отфильтруйте те из них, количество которых не превысило заданного порогового значения.

9.12.3. Пояснение

Twitter API предоставляет прямой доступ к сущностям через поле `entities` в метаданных твитов, как было показано в разделе «Извлечение сущностей из твитов». После извлечения сущностей можно подсчитать, сколько раз каждая появляется в данной коллекции твитов, и отобрать наиболее типичные (как показано в примере 9.13), применив класс `collections.Counter` из стандартной библиотеки Python, который удобно использовать в любых видах частотного анализа на Python. Получив ранжированную коллекцию, останется только от-

фильтровать ее по пороговому значению или другим критериям, чтобы оставить только сущности, представляющие интерес.

Пример 9.13. Поиск самых популярных сущностей в коллекции твитов

```
import twitter
from collections import Counter

def get_common_tweet_entities(statuses, entity_threshold=3):

    # Создать плоский список всех сущностей
    tweet_entities = [ e
                        for status in statuses
                        for entity_type in extract_tweet_entities([status])
                        for e in entity_type
                      ]

    c = Counter(tweet_entities).most_common()

    # Подсчитать количество вхождений
    return [ (k,v)
            for (k,v) in c
              if v >= entity_threshold
            ]

# Пример использования

q = 'CrossFit'

twitter_api = oauth_login()
search_results = twitter_search(twitter_api, q, max_results=100)
common_entities = get_common_tweet_entities(search_results)

print("Most common tweet entities")
print(common_entities)
```

9.13. Вывод результатов частотного анализа в табличной форме

9.13.1. Задача

Требуется вывести результаты частотного анализа в табличной форме, чтобы их проще и удобнее было просматривать.

9.13.2. Решение

Используйте пакет `prettytable`, чтобы создать объект, который позволяет загрузить записи с информацией и отобразить их в виде таблицы со столбцами фиксированной ширины.

9.13.3. Пояснение

Пакет `prettytable` очень прост в использовании и удобен для получения легкочитаемого текстового вывода, который можно скопировать и вставить в любой отчет или текстовый файл (см. пример 9.14). Установить пакет можно обычным для Python способом — выполнив команду `pip install prettytable`. Класс `prettytable.PrettyTable` особенно удобно использовать в сочетании с `collections.Counter` или другими структурами данных, хранящими списки кортежей и способными ранжировать (сортировать) данные.



Если вас интересует вопрос сохранения данных для использования в электронных таблицах, обратитесь к документации с описанием пакета `csv`¹, который входит в состав стандартной библиотеки Python. Но имейте в виду, что в этом пакете существуют известные проблемы (описанные в документации) поддержки Юникода.

Пример 9.14. Вывод результатов частотного анализа в табличной форме

```
from prettytable import PrettyTable

# Получить результаты частотного анализа

twitter_api = oauth_login()
search_results = twitter_search(twitter_api, q, max_results=100)
common_entities = get_common_tweet_entities(search_results)

# Использовать PrettyTable для вывода данных в табличной форме

pt = PrettyTable(field_names=['Entity', 'Count'])
[ pt.add_row(kv) for kv in common_entities ]
pt.align['Entity'], pt.align['Count'] = 'l', 'r' # Установить выравнивание
                                                    # в столбцах

pt._max_width = {'Entity':60, 'Count':10}
print(pt)
```

¹ <http://bit.ly/2KmFsgz>.

9.14. Поиск пользователей, ретвитнувших статус

9.14.1. Задача

Требуется найти всех пользователей, ретвитнувших определенный статус (твит).

9.14.2. Решение

Чтобы найти пользователей, ретвитнувших статус, используйте конечную точку API `GET retweeters/ids`.

9.14.3. Пояснение

Конечная точка API `GET retweeters/ids`¹ возвращает идентификаторы всех пользователей, ретвитнувших указанный статус, однако есть несколько тонких моментов, о которых вам следует знать. В частности, эта конечная точка API возвращает только пользователей, выполнивших ретвит с использованием *встроенного Retweet API*, и не определяет пользователей, которые просто процитировали твит, предварив его комбинацией букв «RT» и добавив ссылку на источник «(via @username)» в конец или использовав какое-то другое распространенное соглашение.

Большинство приложений Twitter (включая пользовательский интерфейс `twitter.com`) используют встроенный Retweet API, но некоторые пользователи могут по-прежнему делиться статусами, выбирая путь «в обход» встроенного API, чтобы вставить свой комментарий или добавить себя в разговор, который иначе они могли бы передавать только как посредники. Например, пользователь может добавить к твиту комментарий «< AWESOME!» (ПРЕВОСХОДНО!), чтобы показать, что разделяет мнение автора, и хотя он будет думать при этом, что сделал *ретвит*, с точки зрения Twitter API такой ретвит будет считаться цитатой. Одна из причин путаницы между ретвитом и цитированием обусловлена тем, что встроенный Retweet API появился в Twitter не так давно. Фактически, понятие ретвита эволюционировало естественным путем, прежде чем в 2010 году команда Twitter отреагировала на это и реализовала встроенный API.

¹ <http://bit.ly/2jRvjNQ>.

Иллюстрация на практическом примере поможет раскрыть эту тонкую техническую деталь: представьте, что @fperez_org опубликовал статус и затем @SocialWebMining ретвитнул его. После этого атрибут `retweet_count` статуса, опубликованного пользователем @fperez_org, получит значение 1, а @SocialWebMining будет иметь твит в своей пользовательской ленте, отмеченный как ретвит статуса пользователя @fperez_org.

Теперь предположим, что @jyeee заметил статус пользователя @fperez_org, просматривая ленту пользователя @SocialWebMining через пользовательский интерфейс `twitter.com` или приложение, такое как TweetDeck¹, и щелкнул на кнопке Retweet (Ретвитнуть). После этого атрибут `retweet_count` статуса пользователя @fperez_org получит значение 2, а @jyeee будет иметь твит в своей пользовательской ленте (а также последний статус @SocialWebMining), отмеченный как ретвит @fperez_org.

Здесь важно понять следующий момент: *с точки зрения любого пользователя, просматривающего ленту @jyeee, промежуточная ссылка — @SocialWebMining — между @fperez_org и @jyeee будет потеряна*. Иначе говоря, источником твита будет признан @fperez_org, независимо от длины цепочки посредников, участвовавших в распространении статуса.

Имея идентификаторы пользователей, ретвитнувших твит, легко можно получить информацию из их профилей, используя конечную точку `API GET users/lookup`. За дополнительными подробностями обращайтесь к разделу «Получение информации из профиля пользователя».

Учитывая, что пример 9.15 может не в полной мере соответствовать вашим желаниям, обязательно прочитайте раздел «Определение автора твита», где описываются дополнительные шаги, помогающие определить распространителей статуса. В нем приводится пример использования регулярного выражения для анализа содержимого твита и определения принадлежности процитированного твита, который может пригодиться при исследовании исторического архива твитов или когда требуется перепроверить авторство.

Пример 9.15. Поиск пользователей, ретвитнувших статус

```
import twitter

twitter_api = oauth_login()

print("""User IDs for retweeters of a tweet by @fperez_org
```

¹ <http://bit.ly/1a1pIbh>.

```

that was retweeted by @SocialWebMining and that @jyeee then retweeted
from @SocialWebMining's timeline\n""")
print(twitter_api.statuses.retweeters.ids(_id=334188056905129984)['ids'])
print(json.dumps(twitter_api.statuses.show(_id=334188056905129984), indent=1))
print()

print("@SocialWeb's retweet of @fperez_org's tweet\n")
print(twitter_api.statuses.retweeters.ids(_id=345723917798866944)['ids'])
print(json.dumps(twitter_api.statuses.show(_id=345723917798866944), indent=1))
print()

print("@jyeee's retweet of @fperez_org's tweet\n")
print(twitter_api.statuses.retweeters.ids(_id=338835939172417537)['ids'])
print(json.dumps(twitter_api.statuses.show(_id=338835939172417537), indent=1))

```



Некоторые пользователи Twitter преднамеренно цитируют твиты, не используя Retweet API, чтобы включить себя в разговор и, возможно, чтобы попасть в число тех, кого ретвитят. Поэтому до сих пор часто можно увидеть ссылки RT и *via*. Фактически некоторые популярные приложения, такие как TweetDeck, включают поддержку функций «Edit & RT» (Править и ретвитнуть) и «Retweet» (Ретвитнуть), как показано на рис. 9.2.



Рис. 9.2. Популярные приложения, такие как TweetDeck, включают поддержку «цитирования» твитов «Edit & RT» (Править и ретвитнуть) и более современной функции «Retweet» (Ретвитнуть)

9.15. Определение автора твита

9.15.1. Задача

Требуется определить автора оригинального твита.

9.15.2. Решение

Используя регулярные выражения, проанализируйте содержимое твита на присутствие соглашений, таких как «RT @SocialWebMining» или «(via @SocialWebMining)».

9.15.3. Пояснение

Анализ результатов, возвращаемых встроенным Retweet API, как было описано в разделе «Поиск пользователей, ретвитнувших статус», не всегда помогает определить автора оригинального твита. Как отмечалось в том рецепте, иногда по каким-то причинам пользователи могут преднамеренно вводить себя в разговор, поэтому может потребоваться проанализировать содержимое самого твита, чтобы определить настоящего автора. Пример 9.16 демонстрирует, как использовать регулярные выражения в Python для поиска пары часто используемых соглашений, принятых до появления встроенного Retweet API и все еще широко используемых в наши дни.

Пример 9.16. Определение автора твита

```
import re
```

```
def get_rt_attributions(tweet):
```

```
    # Регулярное выражение взято с сайта Stack Overflow (http://bit.ly/1821y0J)
```

```
    rt_patterns = re.compile(r"(RT|via)((?:\b\W*@\w+)+)", re.IGNORECASE)
    rt_attributions = []
```

```
    # Проверить твит, был ли он создан конечной точкой /statuses/retweet/:id.
```

```
    # См. https://bit.ly/2BhBEaq
```

```
    if 'retweeted_status' in tweet:
```

```

    attribution = tweet['retweeted_status']['user']['screen_name'].lower()
    rt_attributions.append(attribution)

# Также исследовать твит на присутствие "устаревших" шаблонов цитирования,
# таких как "RT" и "via", которые все еще широко используются по разным
# причинам и иногда могут быть полезными. Дополнительную информацию о ретвитах
# ищите по адресу: https://bit.ly/2piMo6h

try:
    rt_attributions += [
        mention.strip()
        for mention in rt_patterns.findall(tweet['text'])[0][1].
            split()
    ]
except IndexError as e:
    pass

# Отфильтровать дубликаты

return list(set([rta.strip("@").lower() for rta in rt_attributions]))

# Пример использования
twitter_api = oauth_login()

tweet = twitter_api.statuses.show(_id=214746575765913602)
print(get_rt_attributions(tweet))
print()
tweet = twitter_api.statuses.show(_id=345723917798866944)
print(get_rt_attributions(tweet))

```

9.16. Выполнение надежных запросов к Twitter

9.16.1. Задача

В процессе сбора данных для анализа вы обязательно столкнетесь с непредвиденными ошибками HTTP, от превышения ограничений (код ошибки 429) до печально известной «услуга недоступна» (код ошибки 503), которые необходимо обрабатывать в каждом конкретном случае.

9.16.2. Решение

Напишите функцию, играющую роль универсальной обертки для API и поддерживающую логику обработки разных кодов ошибок HTTP.

9.16.3. Пояснение

Несмотря на то что Twitter устанавливает вполне разумные ограничения для большинства применений, они часто оказываются слишком жесткими для упражнений, связанных с анализом данных, поэтому приходится следить за числом выполняемых запросов, делающихся в определенный период времени, а также обрабатывать другие ошибки HTTP, такие как печально известная «услуга недоступна» и другие, вызванные сбоями в сети. В примере 9.17 показан один из подходов, который заключается в определении функции, скрывающей в себе всю необходимую логику и позволяющей писать сценарии, не задумываясь об ограничениях и других ошибках HTTP, как если бы они вообще не существовали.



В разделе «Конструирование удобных вызовов функций» уже было показано, как использовать функцию `functools.partial` из стандартной библиотеки, чтобы упростить создание такой функции-обертки для некоторых ситуаций. Также обязательно ознакомьтесь с полным списком кодов ошибок HTTP, которые может вернуть Twitter¹. В разделе «Получение всех друзей и последователей пользователя» приводится конкретная реализация, которая иллюстрирует применение функции `make_twitter_request`, упрощающей обработку некоторых ошибок HTTP, встречающихся при извлечении данных из Twitter.

Пример 9.17. Выполнение надежных запросов к Twitter

```
import sys
import time
from urllib.error import URLError
from http.client import BadStatusLine
import json
import twitter

def make_twitter_request(twitter_api_func, max_errors=10, *args, **kw):

    # Вложенная вспомогательная функция. Обрабатывает некоторые распространенные
    # ошибки HTTP. Возвращает обновленное значение для wait_period в случае ошибки
    # 500. В случае ошибки превышения ограничения (код ошибки 429) блокирует
    # выполнение, пока ограничение не будет сброшено. Для ошибок 401 и 404,
    # требующих специальной обработки вызывающим кодом, возвращает None.
    def handle_twitter_http_error(e, wait_period=2, sleep_when_rate_limited=True):

        if wait_period > 3600: # Seconds
            print('Too many retries. Quitting.', file=sys.stderr)
```

¹ <http://bit.ly/2rFAjZw>.

```

        raise e

# Коды некоторых распространенных ошибок можно узнать по адресу:
# https://developer.twitter.com/en/docs/basics/response-codes

if e.e.code == 401:
    print('Encountered 401 Error (Not Authorized)', file=sys.stderr)
    return None
elif e.e.code == 404:
    print('Encountered 404 Error (Not Found)', file=sys.stderr)
    return None
elif e.e.code == 429:
    print('Encountered 429 Error (Rate Limit Exceeded)', file=sys.stderr)
    if sleep_when_rate_limited:
        print("Retrying in 15 minutes...ZzZ...", file=sys.stderr)
        sys.stderr.flush()
        time.sleep(60*15 + 5)
        print('...ZzZ...Awake now and trying again.', file=sys.stderr)
        return 2
    else:
        raise e # Caller must handle the rate-limiting issue
elif e.e.code in (500, 502, 503, 504):
    print('Encountered {0} Error. Retrying in {1} seconds'\
        .format(e.e.code, wait_period), file=sys.stderr)
    time.sleep(wait_period)
    wait_period *= 1.5
    return wait_period
else:
    raise e

# Конец вложенной функции

wait_period = 2
error_count = 0

while True:
    try:
        return twitter_api_func(*args, **kw)
    except twitter.api.TwitterHTTPError as e:
        error_count = 0
        wait_period = handle_twitter_http_error(e, wait_period)
        if wait_period is None:
            return
    except URLError as e:
        error_count += 1
        time.sleep(wait_period)
        wait_period *= 1.5
        print("URLError encountered. Continuing.", file=sys.stderr)
        if error_count > max_errors:

```

```
        print("Too many consecutive errors...bailing out.",
              file=sys.stderr)
        raise
    except BadStatusLine as e:
        error_count += 1
        time.sleep(wait_period)
        wait_period *= 1.5
        print("BadStatusLine encountered. Continuing.", file=sys.stderr)
    if error_count > max_errors:
        print("Too many consecutive errors...bailing out.",
              file=sys.stderr)
        raise
```

Пример использования

```
twitter_api = oauth_login()
```

Описание twitter_api.users.lookup см. по адресу: <http://bit.ly/2Gcjfzr>

```
response = make_twitter_request(twitter_api.users.lookup,
                                screen_name="SocialWebMining")
```

```
print(json.dumps(response, indent=1))
```

9.17. Получение информации из профиля пользователя

9.17.1. Задача

Имея идентификаторы или отображаемые имена одного или нескольких пользователей, получить информацию из их профилей.

9.17.2. Решение

Используйте API GET `users/lookup` для получения информации из профилей до 100 пользователей за раз по их идентификаторам или отображаемым именам.

9.17.3. Пояснение

Многие API, такие как GET `friends/ids` и GET `followers/ids`, возвращают ничего не говорящие идентификаторы, которые требуется преобразовать в имена

пользователей или другую информацию из профилей для дальнейшего анализа. Для этих целей Twitter предлагает конечную точку `GET users/lookup`¹. Ей можно передать до 100 идентификаторов или имен пользователей и получить информацию из их профилей, а используя простой шаблон обхода в цикле, с ее помощью можно извлекать большие пакеты данных. Также путем небольшого усложнения логики можно написать единственную функцию, которая способна принимать идентификаторы или имена пользователей, по вашему выбору, в именованных параметрах и преобразовывать их в профили. Пример 9.18 демонстрирует такую функцию, которую можно адаптировать для самых разных целей и обеспечить дополнительную поддержку для случаев, когда может понадобиться получить информацию о пользователях по их идентификаторам.

Пример 9.18. Получение информации из профиля пользователя

```
def get_user_profile(twitter_api, screen_names=None, user_ids=None):

    # Функции должен быть передан один из параметров, screen_name или user_id
    # (для проверки используется логика "Исключающее ИЛИ")
    assert (screen_names != None) != (user_ids != None),
        "Must have screen_names or user_ids, but not both"

    items_to_info = {}

    items = screen_names or user_ids

    while len(items) > 0:

        # Обработать до 100 элементов за раз согласно спецификации
        # API /users/lookup.
        # Подробности ищите по адресу: http://bit.ly/2Gcjfzr.

        items_str = ','.join([str(item) for item in items[:100]])
        items = items[100:]

        if screen_names:
            response = make_twitter_request(twitter_api.users.lookup,
                                           screen_name=items_str)
        else: # user_ids
            response = make_twitter_request(twitter_api.users.lookup,
                                           user_id=items_str)

        for user_info in response:
            if screen_names:
```

¹ <https://bit.ly/2Gcjfzr>.

```
        items_to_info[user_info['screen_name']] = user_info
    else: # user_ids
        items_to_info[user_info['id']] = user_info

    return items_to_info

# Пример использования

twitter_api = oauth_login()

print(get_user_profile(twitter_api,
    screen_names=["SocialWebMining", "ptwobrussell"]))
#print(get_user_profile(twitter_api, user_ids=[132373965]))
```

9.18. Извлечение сущностей твитов из произвольного текста

9.18.1. Задача

Требуется проанализировать произвольный текст и извлечь из него сущности, характерные для твитов, такие как @username, хештеги и адреса URL.

9.18.2. Решение

Для извлечения сущностей твитов из произвольного текста (например, из архива твитов, которые могут содержать сущности, определяемые версией API v1.1) используйте сторонний пакет, такой как `twitter_text`.

9.18.3. Пояснение

Twitter не всегда извлекал сущности из твитов в отдельные поля, но вы легко сможете извлечь их сами, воспользовавшись сторонним пакетом `twitter_text`, как показано в примере 9.19. Установить пакет `twitter_text` можно обычной командой `pip install twitter_text`.

Пример 9.19. Извлечение сущностей твитов из произвольного текста

```
# pip install twitter_text
import twitter_text
```

Пример использования

```
txt = "RT @SocialWebMining Mining 1M+ Tweets About #Syria http://wp.me/p3QjJd-1I"

ex = twitter_text.Extractor(txt)

print("Screen Names:", ex.extract_mentioned_screen_names_with_indices())
print("URLs:", ex.extract_urls_with_indices())
print("Hashtags:", ex.extract_hashtags_with_indices())
```

9.19. Получение всех друзей и последователей пользователя

9.19.1. Задача

Требуется получить список всех друзей или последователей для некоторого (возможно, весьма популярного) пользователя Twitter.

9.19.2. Решение

Используйте функцию `make_twitter_request`, представленную в разделе «Выполнение надежных запросов к Twitter», чтобы упростить извлечение информации по идентификаторам, если число последователей окажется настолько большим, что могут быть превышены установленные ограничения.

9.19.3. Пояснение

Конечные точки `GET friends/ids` и `GET followers/ids` предлагают возможность получить идентификаторы всех друзей или последователей конкретного пользователя. Однако логика получения всех идентификаторов может оказаться весьма нетривиальной, потому что в ответ на каждый запрос API возвращается не более 5000 идентификаторов за раз. Конечно, у большинства пользователей едва ли наберется 5000 друзей или последователей, но у некоторых знаменитостей, часто представляющих большой интерес для анализа, их могут быть сотни тысяч или даже миллионы. Получение всех этих идентификаторов может оказаться сложной задачей, потому что при этом необходимо организовать обход курсора в каждом пакете результатов, а также учесть ненулевую вероятность ошибок HTTP. К счастью, для ее решения

легко можно адаптировать функцию `make_twitter_request` и логику работы с курсором, представленные выше.

Методы, аналогичные представленным в примере 9.20, можно включить в шаблон, описанный в разделе «Получение информации из профиля пользователя», создать надежную функцию и реализовать следующий шаг — получение имен пользователей для всех (или части) идентификаторов. Получаемые результаты мы бы посоветовали сохранять в документоориентированной базе данных, такой как MongoDB (как было показано в разделе «Сохранение данных JSON в MongoDB и доступ к ним»), чтобы не потерять информацию в случае непредвиденного сбоя во время выполнения операции извлечения большого объема данных.



Порой лучше заплатить третьей стороне, например DataSift¹, за возможность более быстрого доступа к некоторым видам данных, таким как полный список всех фолловеров популярных пользователей (например, *@ladygaga*). Прежде чем пытаться извлечь такой огромный объем данных, хотя бы прикиньте, сколько времени это займет, подумайте о вероятных (неожиданных) ошибках, которые могут возникнуть в этом длительном процессе, и оцените возможность получения данных из других источников. То, что стоит денег, может сэкономить массу времени.

Пример 9.20. Получение всех друзей и последователей пользователя

```
from functools import partial
from sys import maxsize as maxint

def get_friends_followers_ids(twitter_api, screen_name=None, user_id=None,
                              friends_limit=maxint, followers_limit=maxint):

    # Функции должен быть передан один из параметров, screen_name или user_id
    # (для проверки используется логика "Исключающее ИЛИ")
    assert (screen_name != None) != (user_id != None),
        "Must have screen_name or user_id, but not both"

    # Сведения о параметрах API ищите по адресам:
    # http://bit.ly/2GcjKJP b http://bit.ly/2rFz90N

    get_friends_ids = partial(make_twitter_request, twitter_api.friends.ids,
                              count=5000)
    get_followers_ids = partial(make_twitter_request, twitter_api.followers.ids,
                                count=5000)
```

¹ <http://bit.ly/1a1pKje>.

```

friends_ids, followers_ids = [], []

for twitter_api_func, limit, ids, label in [
    [get_friends_ids, friends_limit, friends_ids, "friends"],
    [get_followers_ids, followers_limit, followers_ids,
     "followers"]
]:

    if limit == 0: continue

    cursor = -1
    while cursor != 0:

        # Использовать make_twitter_request через частичное связывание
        # параметров
        if screen_name:
            response = twitter_api_func(screen_name=screen_name, cursor=cursor)
        else: # user_id
            response = twitter_api_func(user_id=user_id, cursor=cursor)

        if response is not None:
            ids += response['ids']
            cursor = response['next_cursor']

        print('Fetched {0} total {1} ids for {2}'.format(len(ids),
            label, (user_id or screen_name)), file=sys.stderr)

        # Подумайте о сохранении данных в каждой итерации, чтобы обеспечить
        # дополнительную защиту от исключительных ситуаций

        if len(ids) >= limit or response is None:
            break

    # Сделать что-нибудь полезное с идентификаторами ID, например сохранить
    # на диск
    return friends_ids[:friends_limit], followers_ids[:followers_limit]

# Пример использования

twitter_api = oauth_login()

friends_ids, followers_ids = get_friends_followers_ids(twitter_api,
    screen_name="SocialWebMining",
    friends_limit=10,
    followers_limit=10)

print(friends_ids)
print(followers_ids)

```

9.20. Анализ друзей и последователей пользователя

9.20.1. Задача

Требуется выполнить простой анализ, сравнивающий друзей и последователей пользователя.

9.20.2. Решение

Используйте операции с множествами, такие как пересечение и разность, для анализа друзей и последователей пользователя.

9.20.3. Пояснение

Получив списки всех друзей и последователей пользователя, можно выполнить простейший анализ, используя только идентификаторы и операции с множествами, такие как *пересечение* и *разность*, как показано в примере 9.21.

Операция пересечения с двумя множествами вернет элементы, присутствующие в обоих множествах, а операция разности «вычтет» одно множество из другого, оставив только отличающиеся элементы. Напомню, что операция пересечения является коммутативной, а операция разности — *нет*.¹

В контексте анализа друзей и последователей пересечение двух множеств можно интерпретировать как список пользователей, с кем установлена «взаимная дружба», то есть людей, за которыми следует данный пользователь и которые в свою очередь следуют за данным пользователем. А разность двух множеств можно рассматривать как список последователей, за которыми данный пользователь не следует, или людей, за которыми следует данный пользователь, но которые в свою очередь не следуют за ним, в зависимости от порядка операндов.

При наличии полных списков идентификаторов друзей и последователей эти операции с множествами являются естественными отправными точками и мо-

¹ Коммутативной называется операция, для которой порядок следования операндов не имеет значения: операнды можно переставить местами, как в случае умножения и сложения.

гут служить трамплином для последующего анализа. Например, едва ли есть смысл использовать конечную точку GET users/lookup для получения профилей миллионов последователей. Порой лучше выбрать лиц, с кем у пользователя установлена взаимная дружба (и которые наверняка имеют более сильное сходство), и, используя уже эти идентификаторы, извлекать профили пользователей для последующего анализа.

Пример 9.21. Анализ друзей и последователей пользователя

```
def setwise_friends_followers_analysis(screen_name, friends_ids, followers_ids):

    friends_ids, followers_ids = set(friends_ids), set(followers_ids)

    print('{0} is following {1}'.format(screen_name, len(friends_ids)))

    print('{0} is being followed by {1}'.format(screen_name, len(followers_ids)))

    print('{0} of {1} are not following {2} back'.format(
        len(friends_ids.difference(followers_ids)),
        len(friends_ids), screen_name))

    print('{0} of {1} are not being followed back by {2}'.format(
        len(followers_ids.difference(friends_ids)),
        len(followers_ids), screen_name))

    print('{0} has {1} mutual friends'.format(
        screen_name, len(friends_ids.intersection(followers_ids))))

# Пример использования

screen_name = "ptwobrussell"

twitter_api = oauth_login()

friends_ids, followers_ids = get_friends_followers_ids(twitter_api,
                                                         screen_name=screen_name)
setwise_friends_followers_analysis(screen_name, friends_ids, followers_ids)
```

9.21. Извлечение твитов пользователя

9.21.1. Задача

Требуется получить все недавние твиты пользователя для анализа.

9.21.2. Решение

Используйте конечную точку `GET statuses/user_timeline`, чтобы получить до 3200 самых последних твитов пользователя. Для этого желательно воспользоваться надежной функцией-оберткой, такой как `make_twitter_request` (представлена в разделе «Выполнение надежных запросов к Twitter»), потому что эта серия запросов может превысить установленные ограничения или столкнуться с ошибкой HTTP.

9.21.3. Пояснение

Ленты являются фундаментальным понятием в экосистеме Twitter, и Twitter предлагает удобную конечную точку, позволяющую получить твиты пользователя и реализующую понятие «ленты пользователя». Извлечение твитов пользователя, как показано в примере 9.22, является важной отправной точкой для анализа, потому что твит — это самый основной примитив в экосистеме. Имея большую коллекцию твитов, опубликованных конкретным пользователем, можно получить достаточно полное представление, о чем он говорит (и, соответственно, что его волнует). Кроме того, с архивом, включающим несколько сотен твитов пользователя, можно провести десятки экспериментов, часто почти без обращения к Twitter API. Хранение твитов в определенной коллекции внутри документоориентированной базы данных, такой как MongoDB, является естественным способом хранения и использования данных в экспериментах. Для пользователей, давно зарегистрированных в Twitter, часто бывает любопытно провести анализ лент сообщений, чтобы выяснить, как с течением времени менялись их интересы и настроения.

Пример 9.22. Извлечение твитов пользователя

```
def harvest_user_timeline(twitter_api, screen_name=None, user_id=None,
                          max_results=1000):

    assert (screen_name != None) != (user_id != None), \
        "Must have screen_name or user_id, but not both"

    kw = { # Именованные аргументы для вызова Twitter API
        'count': 200,
        'trim_user': 'true',
        'include_rts': 'true',
        'since_id': 1
    }
```



```

if screen_name:
    kw['screen_name'] = screen_name
else:
    kw['user_id'] = user_id

max_pages = 16
results = []

tweets = make_twitter_request(twitter_api.statuses.user_timeline, **kw)

if tweets is None: # Ошибка 401 (Not Authorized) -- Предотвратить вход в цикл
    tweets = []

results += tweets

print('Fetched {0} tweets'.format(len(tweets)), file=sys.stderr)

page_num = 1

# Многие пользователи Twitter имеют меньше 200 твитов, поэтому нет смысла
# входить в цикл и расходовать драгоценный запрос, если max_results = 200.

# Примечание: аналогичную оптимизацию можно было бы применить внутри цикла
# и сэкономить запросы (например, не выполнять третий запрос, если после
# второго получено 287 твитов из возможных 400). Дело в том, что
# Twitter выполняет некоторую пост-фильтрацию цензурированных или удаленных
# твитов из пакета, поэтому не гарантируется, что число результатов будет
# точно равно 200. Вы можете получить 198 твитов, например, и при этом будут
# доступны для извлечения другие твиты. Если общее число твитов известно
# (можно получить из GET /users/lookup/), тогда можно попробовать
# использовать его для оптимизации.

if max_results == kw['count']:
    page_num = max_pages # Prevent loop entry

while page_num < max_pages and len(tweets) > 0 and len(results) < max_results:

    # Необходимо для обхода ленты сообщений в Twitter API v1.1:
    # получить параметр max-id для следующего запроса.
    # См. http://bit.ly/2L0jwJw.
    kw['max_id'] = min([tweet['id'] for tweet in tweets]) - 1

    tweets = make_twitter_request(twitter_api.statuses.user_timeline, **kw)
    results += tweets

    print('Fetched {0} tweets'.format(len(tweets)), file=sys.stderr)

    page_num += 1

print('Done fetching tweets', file=sys.stderr)

return results[:max_results]

```

Пример использования

```
twitter_api = oauth_login()
tweets = harvest_user_timeline(twitter_api, screen_name="SocialWebMining",
                               max_results=200)
```

Сохранить в MongoDB вызовом `save_to_mongo` или в локальный файл вызовом `save_json`

9.22. Обход графа дружбы

9.22.1. Задача

Требуется получить идентификаторы последователей пользователя, последователей его последователей, последователей последователей этих последователей и т. д. для последующего сетевого анализа — по сути, выполнить обход графа дружбы по связям «следования».

9.22.2. Решение

Используйте прием поиска в ширину для последовательного извлечения информации о дружеских связях, которые можно интерпретировать как граф для сетевого анализа.

9.22.3. Пояснение

Поиск в ширину — распространенный метод исследования графа и один из стандартных способов начать создание нескольких уровней контекста, определяемых отношениями. Имея начальную точку и предельную глубину, поиск в ширину исследует пространство так, что гарантированно вернет все узлы в графе вплоть до заданной глубины, и при этом исследование каждого предыдущего уровня завершится раньше, чем начнется исследование следующего (см. пример 9.23).

Имейте в виду, что при исследовании графов отношений в Twitter можно столкнуться с *суперузлами* — узлами с большим количеством исходящих ребер, — которые легко могут привести к значительным затратам вычислительных ресурсов и расходованию запросов API, которые учитываются механизмом ограничений Twitter. Желательно явно ограничить максимальное число фолловеров, которое может извлекаться для каждого пользователя в графе, хотя

бы на этапе предварительного анализа, чтобы понять, с чем вы столкнулись, присутствуют ли суперузлы в выбранном графе и стоит ли продолжать вкладывать силы и время в решение задачи. Исследование неизвестного графа — сложная (и захватывающая) задача, и существуют разные инструменты, такие как приемы выборки образцов, которые можно задействовать для дальнейшего увеличения эффективности поиска.

Пример 9.23. Обход графа дружбы

```
def crawl_followers(twitter_api, screen_name, limit=1000000, depth=2,
                    **mongo_conn_kw):

    # Получить идентификатор по отображаемому имени пользователя screen_name
    # и работать с идентификаторами для единообразия

    seed_id = str(twitter_api.users.show(screen_name=screen_name)['id'])

    _, next_queue = get_friends_followers_ids(twitter_api, user_id=seed_id,
                                              friends_limit=0, followers_limit=limit)

    # Сохранить отображение seed_id => _follower_ids в MongoDB

    save_to_mongo({'followers' : [ _id for _id in next_queue ]}, 'followers_crawl',
                  '{0}-follower_ids'.format(seed_id), **mongo_conn_kw)

    d = 1
    while d < depth:
        d += 1
        (queue, next_queue) = (next_queue, [])
        for fid in queue:
            _, follower_ids = get_friends_followers_ids(twitter_api, user_id=fid,
                                                        friends_limit=0,
                                                        followers_limit=limit)

            # Сохранить отображение fid => follower_ids в MongoDB

            save_to_mongo({'followers' : [ _id for _id in follower_ids ]},
                          'followers_crawl', '{0}-follower_ids'.format(fid))

            next_queue += follower_ids

# Пример использования

screen_name = "timoreilly"

twitter_api = oauth_login()

crawl_followers(twitter_api, screen_name, depth=1, limit=10)
```

9.23. Анализ содержимого твитов

9.23.1. Задача

Требуется выполнить краткий анализ твитов в имеющейся коллекции, чтобы получить представление о характере обсуждения и понятиях, передаваемых в самих твитах.

9.23.2. Решение

Вычислите простейшие статистики, такие как лексическое разнообразие и среднее число слов в твите, чтобы определить природу языка как первый шаг к пониманию, о чем ведется речь.

9.23.3. Обсуждение

В дополнение к выявлению сущностей твитов и простому частотному анализу для определения часто используемых слов также можно получить оценку *лексического разнообразия* твитов и вычислить другие простые статистики, такие как среднее число слов в твите, чтобы полнее понять имеющиеся данные (см. пример 9.24). Лексическое разнообразие — это простая статистика, которая определяется как отношение числа уникальных слов к общему числу слов в корпусе; по определению лексическое разнообразие 1,0 означает, что все слова в корпусе уникальны, тогда как лексическое разнообразие, приближающееся к 0,0, подразумевает большое число повторяющихся слов.

Лексическое разнообразие может интерпретироваться по-разному, в зависимости от контекста. Например, в литературе оценки лексического разнообразия можно использовать как меру богатства и выразительности языка при сравнении авторов. Изучение лексического разнообразия обычно не является конечной целью, но этот шаг помогает получить первую полезную информацию (обычно в сочетании с частотным анализом), которую можно использовать для планирования следующих этапов.

В контексте Twitter лексическое разнообразие тоже можно интерпретировать аналогичным образом и использовать для сравнения пользователей, но, кроме этого, оно может многое сказать об относительном разнообразии обсуждаемой темы, как в случае, когда кто-то говорит только об определенной технологии,

а другой публикует рассуждения по широкому спектру тем. В коллекции твитов нескольких авторов на одну и ту же тему (как это обычно бывает, когда коллекция получена обращением к Search API или Streaming API) неожиданно низкое лексическое разнообразие *может* также указывать на возникновение эффекта «группового мышления». Другая возможная ситуация — большое число ретвитов, когда одна и та же информация повторяется многократно. Как и в любом другом анализе, никакие статистики не должны интерпретироваться без учета контекста.

Пример 9.24. Анализ содержимого твитов

```
def analyze_tweet_content(statuses):

    if len(statuses) == 0:
        print("No statuses to analyze")
        return

    # Вложенная вспомогательная функция для вычисления
    # лексического разнообразия
    def lexical_diversity(tokens):
        return 1.0*len(set(tokens))/len(tokens)

    # Вложенная вспомогательная функция для вычисления
    # среднего числа слов в твите
    def average_words(statuses):
        total_words = sum([ len(s.split()) for s in statuses ])
        return 1.0*total_words/len(statuses)

    status_texts = [ status['text'] for status in statuses ]
    screen_names, hashtags, urls, media, _ = extract_tweet_entities(statuses)

    # Получить коллекцию всех слов из всех твитов
    words = [ w
        for t in status_texts
        for w in t.split() ]

    print("Lexical diversity (words):", lexical_diversity(words))
    print("Lexical diversity (screen names):", lexical_diversity(screen_names))
    print("Lexical diversity (hashtags):", lexical_diversity(hashtags))
    print("Average words per tweet:", average_words(status_texts))

# Пример использования

q = 'CrossFit'
twitter_api = oauth_login()
search_results = twitter_search(twitter_api, q)

analyze_tweet_content(search_results)
```

9.24. Обобщение целевых ссылок

9.24.1. Задача

Требуется получить общее представление, о чем говорится в целевой ссылке, такой как адрес URL, извлеченный из сущностей твита, чтобы понять природу твита или интересы пользователя Twitter.

9.24.2. Решение

Обобщите содержимое по адресу в URL до нескольких предложений, которые проще просмотреть (или проанализировать каким-то другим способом), чем читать веб-страницу целиком.

9.24.3. Пояснение

Когда дело доходит до попытки понять текст на человеческом языке, опубликованный на веб-странице, ваши возможности ограничены только вашим воображением. Пример 9.25 можно рассматривать как попытку представить шаблон обработки и обобщения содержимого в более краткую форму, более удобную для ознакомления или анализа альтернативными способами. Проще говоря, он показывает, как получить веб-страницу, выделить из нее основное содержимое (отфильтровав массу текста в верхнем и нижнем колонтитулах и боковых панелях и т. д.), удалить разметку HTML, которая может оставаться в основном содержимом, и использовать простой прием обобщения для получения наиболее значимых предложений из содержимого.

Демонстрируемый прием основывается на предпосылке, что наиболее важные предложения могут служить хорошим обобщением, если следуют в хронологическом порядке, и такие предложения идентифицируются часто встречающимися словами, располагающимися по соседству друг с другом. Несмотря на довольно грубые допущения, эта форма обобщения показывает удивительно хорошие результаты для грамотно написанного текста.

Пример 9.25. Обобщение целевых ссылок

```
import sys
import json
import nltk
```

```

import numpy
import requests
from boilerpipe.extract import Extractor

def summarize(url=None, html=None, n=100, cluster_threshold=5, top_sentences=5):

    # Адаптированная версия функции из статьи Х. П. Луна (H. P. Luhn)
    # "The Automatic Creation of Literature Abstracts"
    #
    # Параметры:
    # * n - число рассматриваемых слов
    # * cluster_threshold - пороговое расстояние между словами
    # * top_sentences - число возвращаемых предложений

    # Начало вложенной вспомогательной функции
    def score_sentences(sentences, important_words):
        scores = []
        sentence_idx = -1

        for s in [nltk.tokenize.word_tokenize(s) for s in sentences]:

            sentence_idx += 1
            word_idx = []

            # Для каждого слова в списке...
            for w in important_words:
                try:
                    # Определить индексы важных слов в каждом предложении
                    word_idx.append(s.index(w))
                except ValueError as e: # w отсутствует в данном предложении
                    pass

            word_idx.sort()

            # Некоторые предложения могут не содержать некоторых важных слов
            if len(word_idx) == 0: continue

            # Используя позиции, определить кластеры на основе предельного
            # расстояния для любых двух соседствующих слов

            clusters = []
            cluster = [word_idx[0]]
            i = 1
            while i < len(word_idx):
                if word_idx[i] - word_idx[i - 1] < cluster_threshold:
                    cluster.append(word_idx[i])
                else:
                    clusters.append(cluster[:])
                    cluster = [word_idx[i]]
                i += 1

```

```

clusters.append(cluster)

# Оценить каждый кластер. Выбрать максимальную оценку кластера
# в качестве оценки для всего предложения.

max_cluster_score = 0
for c in clusters:
    significant_words_in_cluster = len(c)
    total_words_in_cluster = c[-1] - c[0] + 1
    score = 1.0 * significant_words_in_cluster \
        * significant_words_in_cluster / total_words_in_cluster

    if score > max_cluster_score:
        max_cluster_score = score

scores.append((sentence_idx, score))

return scores

# Конец вложенной вспомогательной функции

extractor = Extractor(extractor='ArticleExtractor', url=url, html=html)

# В общем случае возможно, что эта "чистая страница" будет щедро сдобрена
# ничего не значащими словами. Можете не согласиться с этим. К счастью,
# алгоритм обобщения по своей природе успешно справляется с этим шумом.

txt = extractor.getText()

sentences = [s for s in nltk.tokenize.sent_tokenize(txt)]
normalized_sentences = [s.lower() for s in sentences]

words = [w.lower() for sentence in normalized_sentences for w in
          nltk.tokenize.word_tokenize(sentence)]

fdist = nltk.FreqDist(words)

top_n_words = [w[0] for w in fdist.items()
               if w[0] not in nltk.corpus.stopwords.words('english')][:n]

scored_sentences = score_sentences(normalized_sentences, top_n_words)

# Обобщение, подход 1:
# Отбросить неважные предложения, используя как фильтр среднюю оценку
# плюс долю стандартного отклонения

avg = numpy.mean([s[1] for s in scored_sentences])
std = numpy.std([s[1] for s in scored_sentences])
mean_scored = [(sent_idx, score) for (sent_idx, score) in scored_sentences
               if score > avg + 0.5 * std]

# Обобщение, подход 2:
# Другое решение -- вернуть только N предложений с самыми высокими оценками

```



```

top_n_scored = sorted(scored_sentences, key=lambda s: s[1])[-top_sentences:]
top_n_scored = sorted(top_n_scored, key=lambda s: s[0])

# Дополнить объект документа обобщениями

return dict(top_n_summary=[sentences[idx] for (idx, score) in top_n_scored],
            mean_scored_summary=[sentences[idx] for (idx, score)
                                in mean_scored])

# Пример использования

sample_url = 'http://radar.oreilly.com/2013/06/phishing-in-facebooks-pond.html'
summary = summarize(url=sample_url)

# Как вариант, можно передать разметку. Иногда это решение можно использовать
# для преодоления мистических ошибок, возникающих в urllib2.BadStatusLine.
# Вот как это можно сделать:

# sample_html = requests.get(sample_url).text
# summary = summarize(html=sample_html)

print("-----")
print("          'Top N Summary'")
print("-----")
print(" ".join(summary['top_n_summary']))
print()
print()
print("-----")
print("          'Mean Scored' Summary")
print("-----")
print(" ".join(summary['mean_scored_summary']))

```

9.25. Анализ избранных твитов пользователя

9.25.1. Задача

Требуется узнать больше о том, что волнует человека, изучив твиты, которые он отметил как избранные.

9.25.2. Решение

Используйте конечную точку GET favorites/list, чтобы получить твиты, отмеченные пользователем как избранные, и затем примените к ним приемы для определения, извлечения и подсчета сущностей, характеризующих содержимое твитов.

9.25.3. Пояснение

Не все пользуются функцией добавления в закладки для отметки избранных твитов, поэтому ее нельзя считать надежным способом определения твитов, содержимое которых представляет интерес; но если вам повезет встретить пользователя, который взял в привычку добавлять заинтересовавшие его твиты в закладки, знайте, что вы нашли сокровищницу с курируемым контентом. В примере 9.26 представлен анализ, основанный на предыдущих рецептах конструирования таблицы сущностей, однако вы можете применить более продвинутые методы к содержимому твитов. Например, твиты можно разбить на темы по их содержимому, проанализировать, как менялись предпочтения человека с течением времени, или построить график, демонстрирующий, когда и как часто человек отмечал твиты как избранные.

Имейте в виду, что кроме избранных твитов, ретвиты тоже являются достойными кандидатами для анализа. Например, весьма поучительно было бы проанализировать такие модели поведения, как склонность и частоту пользователя публиковать ретвиты, добавлять твиты в закладки или и то и другое.

Пример 9.26. Анализ избранных твитов пользователя

```
def analyze_favorites(twitter_api, screen_name, entity_threshold=2):

    # Можно выбрать и больше 200 твитов, организовав обход курсора, как показано
    # в других рецептах, но 200 -- достаточно представительная выборка
    favs = twitter_api.favorites.list(screen_name=screen_name, count=200)
    print("Number of favorites:", len(favs))

    # Выбрать из содержимого некоторые из наиболее распространенных сущностей
    common_entities = get_common_tweet_entities(favs,
                                                entity_threshold=entity_threshold)

    # Вывести результаты в табличной форме, используя PrettyTable

    pt = PrettyTable(field_names=['Entity', 'Count'])
    [ pt.add_row(kv) for kv in common_entities ]
    pt.align['Entity'], pt.align['Count'] = 'l', 'r' # Set column alignment

    print()
    print("Common entities in favorites...")
    print(pt)

    # Вывести некоторые другие статистики
    print()
    print("Some statistics about the content of the favorites...")
    print()
```

```
analyze_tweet_content(favs)

# Можно также добавить анализ содержимого по ссылке с обобщением
# и многое другое

# Пример использования

twitter_api = oauth_login()
analyze_favorites(twitter_api, "ptwobrussell")
```

9.26. Заключительные замечания

Этот сборник содержит очень небольшую коллекцию, если сравнивать его со сборниками, содержащими сотни или даже тысячи рецептов для анализа данных из Twitter, и тем не менее мы надеемся, что он дал вам хорошую основу и набор идей, на которые вы сможете опереться и адаптировать их для решения многих интересных задач. Анализ данных из Twitter (и большинства других социальных сетей) имеет широкие, мощные и (что, пожалуй, самое важное) интересные перспективы!



Запросы на дополнительные рецепты (а также на расширения представленных здесь рецептов) приветствуются и с радостью будут приняты. Смело забирайте исходный код примеров для этой книги из репозитория GitHub¹, присылайте свои рецепты в формате блокнота Jupyter Notebook для этой главы и оставляйте свои запросы на создание новых рецептов! Мы надеемся, что эта коллекция будет расширяться, станет ценной отправной точкой для хакеров, занимающихся анализом данных из социальных сетей, и соберет вокруг себя активное сообщество участников.

9.27. Упражнения

- Ознакомьтесь детальнее с программным интерфейсом Twitter Platform API². Удивило ли вас наличие (или отсутствие) обнаруженных программных интерфейсов?
- Проанализируйте все твиты, которые вы когда-либо ретвитили. Удивил ли вас сам факт, что вы ретвитили их, или, может быть, вас удивило, как менялись ваши интересы с течением времени?

¹ <http://bit.ly/Mining-the-Social-Web-3e>.

² <http://bit.ly/1a1kSKQ>.

- Сопоставьте твиты, ретвитнутые вами, с твитами, которые вы писали сами. Насколько похожие темы в них обсуждаются?
- Напишите рецепт, загружающий граф дружбы из MongoDB, отображающий его в графическом интерфейсе с помощью NetworkX и использующий для его построения один из встроенных алгоритмов NetworkX, таких как оценка центральности или анализ клик (групп). Перед выполнением этого упражнения может быть полезно еще раз прочитать главу 8, где приводится обзор возможностей NetworkX, который может вам пригодиться до завершения этого упражнения.
- Напишите рецепт, адаптирующий приемы визуализации из предыдущих глав для отображения данных из Twitter. Например, попробуйте отобразить граф дружбы, адаптируйте методы вывода графиков или гистограмм в Jupyter Notebook для визуализации закономерностей, выявленных в твитах, или трендов для конкретного пользователя; можно также попробовать сконструировать облако тегов (такое, как Word Cloud Layout¹) из содержимого твитов.
- Напишите рецепт для выявления последователей, за которыми вы сами не следуете, но могли бы следовать, основываясь на содержимом их твитов. Некоторые меры сходства, которые можно использовать для начала, были представлены в разделе «Измерение степени сходства» главы 4.
- Напишите рецепт, вычисляющий сходство двух пользователей на основе содержимого их твитов.
- Прочитайте еще раз описание Twitter Lists API, в особенности описание конечных точек `/lists/list`² и `/lists/memberships`³, которые возвращают списки, на которые подписан пользователь, и списки, в которые он был добавлен другими пользователями соответственно. Что можно узнать о пользователях по спискам, на которые они подписаны и/или были добавлены другими пользователями?
- Попробуйте применить к твитам методы обработки текста на естественном языке. В Университете Карнеги — Меллона был создан проект «Twitter NLP and part-of-speech tagging»⁴, который послужит вам отличной отправной точкой.

¹ <http://bit.ly/1a1n5pO>.

² <http://bit.ly/2L0fSzd>.

³ <http://bit.ly/2rE3mwD>.

⁴ <http://bit.ly/1a1n84Y>.

- Следуя за большим количеством учетных записей в Twitter, почти невозможно уследить за их активностью. Напишите алгоритм, ранжирующий твиты в вашей ленте по важности, а не в порядке их появления. Сможете ли вы эффективно отфильтровать шум и получить более полезный сигнал? Сможете ли вы получить значимый обзор лучших твитов за сутки, исходя из своих личных интересов?
- Начните собирать коллекцию рецептов для других социальных сетей, таких как Facebook или LinkedIn.

9.28. Онлайн-ресурсы

Ниже приводится список ссылок, упоминавшихся в этой главе, которые могут оказаться полезными для вас:

- BSON¹.
- Репозиторий GitHub проекта d3-cloud².
- Экспоненциальное затухание³.
- Механизм агрегирования данных в MongoDB⁴.
- OAuth 2.0⁵.
- Документация с описанием Twitter API⁶.
- Коды HTTP-ошибок Twitter API⁷.
- Проект «Twitter NLP and part-of-speech tagging»⁸.
- Twitter Streaming API⁹.
- Служба Where On Earth (WOE)¹⁰.

¹ <http://bit.ly/1a1pG34>.

² <http://bit.ly/1a1n5pO>.

³ <http://bit.ly/1a1pHEe>.

⁴ <http://bit.ly/1a1pGjv>.

⁵ <http://oauth.net/2/>.

⁶ <http://bit.ly/1a1kSKQ>.

⁷ <http://bit.ly/2rFAjZw>.

⁸ <http://bit.ly/1a1n84Y>.

⁹ <http://bit.ly/2rDU17W>.

¹⁰ <http://bit.ly/2jVIcXo>.

Часть III

ПРИЛОЖЕНИЯ

В приложениях к этой книге представлены некоторые сопутствующие материалы, охватывающие большую часть предыдущих глав:

- В приложении А дается краткий обзор технологии, обеспечивающей работу виртуальной машины с примерами для этой книги, а также кратко обсуждается назначение и область применения виртуальных машин.
- Приложение Б содержит краткое описание отраслевого протокола Open Authorization (OAuth), который обеспечивает доступ к данным практически с любого известного социального веб-сайта через его программный интерфейс.
- В приложении В приводится очень короткий пример, демонстрирующий некоторые распространенные идиомы программирования на Python, с которыми вы неизбежно встретитесь в исходном коде примеров для этой книги; в нем подчеркиваются некоторые особенности Jupyter Notebook, знание которых может вам пригодиться.

А Информация о виртуальной машине с примерами для этой книги

Для каждой главы в этой книге имеется соответствующий блокнот Jupyter Notebook с примерами, точно так же имеются блокноты Jupyter Notebook для каждого приложения. Все блокноты, независимо от целей, хранятся в репозитории GitHub книги¹. Данное приложение, которое вы читаете в печатном издании, служит ссылкой на блокнот Jupyter Notebook с пошаговыми инструкциями по установке и настройке виртуальной машины с примерами, выполняющейся в контейнере Docker.

Мы настоятельно советуем использовать образ Docker, сопровождающий книгу, в качестве среды разработки вместо версии Python, возможно, установленной у вас. Установка и настройка Jupyter Notebook, а также всех зависимостей, необходимых для научных вычислений, сопряжена с большими сложностями. Различия в версиях сторонних пакетов для Python, которые используются на протяжении всей книги, и необходимость поддержки пользователей, работающих на разных платформах, только усугубляют сложности, возникающие при установке и настройке базовой среды разработки.

В репозитории GitHub² вы найдете самые свежие инструкции, включая описание порядка установки Docker и запуска в контейнере образа, содержащего все необходимые программные компоненты, используемые в книге. *Даже*

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² Там же.

если вы обладаете богатым опытом использования инструментов разработки на Python, вы все равно сможете сэкономить время и силы, воспользовавшись виртуальной машиной для этой книги, при первом ее чтении. Попробуйте, и вы не пожалеете.



Блокнот для Jupyter Notebook, соответствующий приложению А «Информация о виртуальной машине с примерами для этой книги»¹, находится в репозитории GitHub² этой книги и содержит все необходимые пошаговые инструкции.

¹ <http://bit.ly/2H0nbUu>.

² <http://bit.ly/Mining-the-Social-Web-3E>.

Б

Основы OAuth

Для каждой главы в этой книге имеется соответствующий блокнот Jupyter Notebook с примерами, точно так же имеются блокноты Jupyter Notebook для каждого приложения. Все блокноты, независимо от целей, хранятся в репозитории GitHub книги¹. Данное приложение, которое вы читаете в печатном издании, служит ссылкой на блокнот Jupyter Notebook с примером кода, демонстрирующим интерактивный процесс авторизации пользователя с привлечением OAuth, который вам придется использовать, если вы решите реализовать приложение для пользователей.

В оставшейся части раздела дается краткое описание основных особенностей протокола OAuth. Примеры использования OAuth для доступа к популярным веб-сайтам, таким как Twitter, Facebook и LinkedIn, можно найти в соответствующем блокноте Jupyter Notebook, доступном в репозитории с исходным кодом примеров для этой книги.



Как и для других приложений, для приложения Б «Основы OAuth» тоже имеется блокнот Jupyter Notebook², который можно просматривать онлайн.

Обзор

Название OAuth означает «Open Authorization» (открытая авторизация). Этот протокол дает пользователям возможность *авторизовать* приложение для

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² <http://bit.ly/2xnKpUX>.

доступа к данным в их учетных записях посредством API без передачи конфиденциальных учетных данных, таких как имя пользователя и пароль. Несмотря на то что здесь протокол OAuth рассматривается в контексте социальных сетей, он имеет более широкую область применения — в любых контекстах, где пользователю может потребоваться авторизовать приложение для выполнения некоторых действий от его имени. Как правило, пользователи могут управлять уровнем доступа стороннего приложения (в зависимости от детальности API, реализованного поставщиком услуг) и отзывать разрешение в любой момент. Например, в Facebook реализована чрезвычайно детализированная схема разрешений, позволяющая пользователям давать сторонним приложениям доступ к узко ограниченным разделам своей учетной записи.

Учитывая почти повсеместную популярность таких платформ, как Twitter, Facebook, LinkedIn и Instagram, а также пользу, которую несут сторонние приложения, разработанные на этих социальных платформах, неудивительно, что эти платформы приняли на вооружение протокол OAuth как универсальное средство доступа к ним. Так же как в случае с любым стандартом или протоколом, реализации OAuth в разных социальных сетях могут отличаться в зависимости от выбранной версии стандарта, а кроме того, в некоторых реализациях имеются свои уникальные особенности. В оставшейся части раздела дается общий обзор версий OAuth 1.0a, определяемой в RFC 5849¹, и OAuth 2.0, определяемой в RFC 6749², с которыми вы столкнетесь при анализе данных из социальных сетей и в других ситуациях, связанных с организацией взаимодействий с некоторой платформой через ее программный интерфейс.

OAuth 1.0a

Версия OAuth 1.0³ определяет протокол, который позволяет веб-клиенту получить доступ к защищенному ресурсу на сервере, и подробно описана в руководстве «OAuth 1.0 Guide»⁴. Как вы уже знаете, причина появления этого протокола обусловлена необходимостью избежать проблем, связанных с передачей пользователями (владельцами ресурсов) паролей веб-приложений.

¹ <http://bit.ly/1a1pWio>.

² <http://bit.ly/1a1pWiz>.

³ На протяжении всего обсуждения под версией «OAuth 1.0» на самом деле будет подразумеваться версия «OAuth 1.0a», потому что версия OAuth 1.0 признана устаревшей и в настоящее время наибольшее распространение получили реализации ее ревизии «a».

⁴ <http://bit.ly/1a1pYHe>.

И несмотря на довольно узкую область определения, он прекрасно справляется со своей задачей. Как оказывается, одной из основных претензий разработчиков к OAuth 1.0, которые первоначально препятствовали внедрению протокола, была трудоемкость реализации из-за разнообразных деталей, связанных с шифрованием (таких, как генерирование подписи HMAC¹), учитывая, что OAuth 1.0 не предполагает обмен учетными данными через безопасное соединение SSL с использованием протокола HTTPS. Иначе говоря, OAuth 1.0 предполагает использование механизмов криптографии, чтобы гарантировать безопасную передачу данных по проводам.

Несмотря на неформальный характер нашей дискуссии, для вас может быть важно знать, что на языке OAuth приложение, запрашивающее доступ, часто называют *клиентом* (или иногда *получателем*), социальный веб-сайт или службу, где хранятся защищенные ресурсы, — *сервером* (или иногда *поставщиком услуг*), а пользователя, который дает доступ, — *владельцем ресурса*. Поскольку в процессе участвуют три стороны, последовательность пересылки сообщений между ними часто называют *трехсторонним процессом*, или (в разговорной речи) «танцем OAuth». Несмотря на некоторую замысловатость деталей реализации и поддержки безопасности, танец OAuth включает лишь несколько основных шагов, которые позволяют клиентскому приложению получить доступ от имени его владельца к защищенному ресурсу, хранящемуся у поставщика услуг:

1. Клиент получает неавторизованный токен запроса от поставщика услуг.
2. Владелец ресурса авторизует токен запроса.
3. Клиент обменивает токен запроса на токен доступа.
4. Клиент использует токен доступа для обращения к защищенному ресурсу от имени его владельца.

С точки зрения конкретных учетных данных клиент начинает танец OAuth, имея *ключ* и *секрет получателя*, а завершает его с *токеном доступа* и *секретом токена доступа*, которые использует для обращения к защищенному ресурсу. То есть, учитывая все сказанное выше, OAuth 1.0 дает клиентским приложениям возможность безопасно получить авторизацию от владельца ресурса для доступа к ресурсам в учетной записи, хранящимся у поставщика услуг, и несмотря на некоторые, порой утомительные детали реализации, он предоставляет общепринятый протокол, прекрасно справляющийся с этой задачей. Вполне вероятно, что OAuth 1.0 еще будет поддерживаться некоторое время.

¹ <http://bit.ly/1a1pZe1>.



В своей статье «Introduction to OAuth (in Plain English)»¹ Роб Собер (Rob Sober) наглядно показывает, как пользователь (будучи владельцем ресурса) может авторизовать службу коротких ссылок, такую как bit.ly (выступающую в роли клиента) для автоматической публикации ссылок в Twitter (поставщик услуг). Вам определенно стоит познакомиться с основными понятиями, представленными в этой статье.

OAuth 2.0

Если OAuth 1.0 определяет хотя и узкий, но полезный процесс авторизации веб-приложений, то версия OAuth 2.0 изначально разрабатывалась с целью упростить ее реализацию для разработчиков веб-приложений, полностью полагаясь на SSL в вопросах безопасности, и обеспечить гораздо более широкий круг применений, в который входит поддержка и мобильных приложений, и корпоративных приложений, и даже приложений из мира «интернета вещей», выполняющихся на бытовых устройствах, которые могут появиться у вас дома.

Facebook стала первой из социальных сетей, запланировавших миграцию на ранние версии OAuth 2.0 еще в 2011 году², и быстро реализовала платформу, основанную исключительно на части стандарта OAuth 2.0. Даже притом, что стандартная аутентификация пользователей в Twitter все еще основывается на OAuth 1.0a, в начале 2013 года в этой социальной сети была реализована аутентификация приложений³, моделирующая процесс «Client Credentials Grant»⁴ из стандарта OAuth 2.0. Как можете убедиться сами, реакция была довольно неоднозначной, в том смысле, что не все социальные сети поспешили реализовать стандарт OAuth 2.0 сразу после его появления.

До сих пор остается неясным, станет ли OAuth 2.0 новым отраслевым стандартом, как предполагалось первоначально. В одной популярной статье в блоге, озаглавленной как «OAuth 2.0 and the Road to Hell»⁵ (и в ее обсуждении на сайте Hacker News⁶), рассматриваются многочисленные проблемы этого стандарта, и она определенно заслуживает вашего внимания. Статья была написана Эраном Хаммером (Eran Hammer), который в середине 2012 года оставил пост ведущего автора и редактора стандарта OAuth 2.0 после нескольких лет работы

¹ <http://bit.ly/1a1pXD7>.

² <http://bit.ly/1a1pYa9>.

³ <http://bit.ly/2GZEa9a>.

⁴ <http://bit.ly/1a1q3KT>.

⁵ <http://bit.ly/2Jege7m>.

⁶ <http://bit.ly/1a1q2Xg>.

над ним. Похоже, что «разработка комитетом»¹ решений бесконечного числа крупных проблем корпоративного уровня задушила энтузиазм рабочей группы. И хотя стандарт был опубликован в конце 2012 года, пока неясно, является ли он действительным стандартом или всего лишь его проектом. К счастью, в предыдущие годы появилось множество отличных фреймворков OAuth, устранивших большинство сложностей реализации OAuth 1.0, связанных с доступом к API, и разработчики продолжили вводить новшества, несмотря на первоначальные препятствия, имевшиеся в OAuth 1.0. Например, как было показано в первых главах этой книги, благодаря использованию пакетов для Python нам не пришлось вникать или реализовать какие-то сложные детали, связанные с особенностями поддержки OAuth 1.0a; нам понадобилось лишь понять основной принцип работы. Несмотря на некоторый паралич и «добрые намерения», связанные с OAuth 2.0, совершенно очевидно, что некоторые из его процессов определены достаточно четко, чтобы крупные социальные сети могли взять его на вооружение.

Как вы теперь знаете, в отличие от реализаций OAuth 1.0, состоящих из довольно жестко ограниченной последовательности шагов, реализации OAuth 2.0 могут иметь некоторые отличия в зависимости от конкретного применения. Однако на самом деле типичный процесс OAuth 2.0, использующий преимущества SSL и, по сути, состоящий из нескольких переадресаций, мало отличается от вышеупомянутой последовательности шагов в процессе OAuth 1.0. Например, аутентификация приложений² в Twitter включает в себя лишь немногим больше, чем обмен ключа и секрета получателя на токен доступа через защищенное соединение SSL. Отметим еще раз, что реализации будут отличаться в зависимости от конкретного применения, и если вам интересны детали, обращайтесь к разделу 4 стандарта OAuth 2.0³, который написан хотя и не самым легким, но вполне понятным языком. Если вы решитесь прочитать его, просто имейте в виду, что некоторые термины в OAuth 1.0 и OAuth 2.0 немного отличаются, поэтому будет проще сосредоточиться сначала на одном стандарте, а потом на другом, вместо того чтобы изучать их одновременно.



В главе 9 книги *Programming Social Applications*⁴ (O'Reilly) Джонатана Лебланка (Jonathan LeBlanc) вы найдете интересное обсуждение OAuth 1.0 и OAuth 2.0 в контексте создания социальных веб-приложений.

¹ Имеется в виду неправильная организация разработки без централизованного управления. — *Примеч. пер.*

² <http://bit.ly/2GZEa9a>.

³ <http://bit.ly/1a1q3uv>.

⁴ <http://oreil.ly/18I8Yтc>.

Особенности OAuth и базовых реализаций OAuth 1.0 и OAuth 2.0, скорее всего, не будут иметь большого значения для вас как исследователя социальных сетей. Это обсуждение специально было построено так, чтобы дать вам общее представление и понимание основных идей, а также показать некоторые отправные точки для дальнейшего изучения и исследования, если вы пожелаете заняться этим. Как вы уже, наверное, поняли, дьявол действительно кроется в деталях. К счастью, появилось множество хороших сторонних библиотек, в значительной степени избавляющих от необходимости знать все тонкости, по большому счету ненужные в повседневной работе, хотя иногда это знание может пригодиться. Онлайн-код для этого приложения включает процессы для OAuth 1.0 и OAuth 2.0, и вы с его помощью можете узнать столько подробностей, сколько захотите.

В

Советы и рекомендации для Python и Jupyter Notebook

Для каждой главы в этой книге имеется соответствующий блокнот Jupyter Notebook с примерами, точно так же имеются блокноты Jupyter Notebook для каждого приложения. Это приложение, как и приложение А, которое вы читаете в печатном издании, служит ссылкой на блокнот Jupyter Notebook в репозитории GitHub¹ с исходным кодом примеров для этой книги, который содержит коллекцию идиом программирования на языке Python, а также полезные советы по использованию Jupyter Notebook.



Блокнот для Jupyter Notebook, соответствующий приложению В «Советы и рекомендации для Python и Jupyter Notebook»², содержит дополнительные примеры распространенных идиом программирования на языке Python, которые могут оказаться для вас особенно актуальными при работе с этой книгой. Там же вы найдете несколько полезных советов по использованию Jupyter Notebook, которые помогут вам сэкономить время и силы.

Несмотря на то что Python нередко называют «выполняемым псевдокодом», краткий обзор Python как универсального языка программирования может быть полезным для читателей, только начинающих осваивать его. Если вам кажется, что было бы целесообразно познакомиться с основами программирования на Python, найдите время и прочитайте разделы с 1 по 8 руководства «Python Tutorial»³. Это станет по-настоящему выгодным вложением ваших сил и поможет получить максимальное удовольствие от этой книги.

¹ <http://bit.ly/Mining-the-Social-Web-3E>.

² <http://bit.ly/2IUWWVm>.

³ <http://bit.ly/2LHjGph>.

Об авторах

Мэтью Рассел (Matthew Russell, @ptwobrussell) — ведущий специалист из Мидл Теннесси (Middle Tennessee). На работе старается быть лидером, помогает другим становиться лидерами и создает высокоэффективные команды для решения сложных задач. Вне работы размышляет о реальности, практикует ярко выраженный индивидуализм и готовится к зомби-апокалипсису и восстанию машин.

Михаил Классен (Mikhail Klassen, @MikhailKlassen) — главный специалист по обработке и анализу данных в Paladin AI, начинающей компании, создающей адаптивные технологии обучения. Имеет степень кандидата астрофизики, полученную в Университете Макмастера (McMaster University), и степень бакалавра в области прикладной физики, полученную в Колумбийском университете (Columbia University). Михаил увлекается проблемами искусственного интеллекта и применением инструментов для анализа данных в благих целях. Когда он не работает, то обычно читает или путешествует.

Об обложке

На обложке книги изображен сурок (*Marmota monax*), также известный как лесной сурок (это название — *woodchuck* — является производным от *wuchak*; так называли сурка индейцы из племени алгонкинов). Сурки у многих ассоциируются с известным в США и Канаде праздником «День сурка», который отмечается 2 февраля. Согласно народной примете, если в этот день сурок выйдет из норы и увидит свою тень, зима продлится еще шесть недель. Сторонники этого поверья утверждают, что точность такого прогноза составляет от 75 до 90 процентов. Во многих городах есть свои знаменитые сурки-предсказатели, в том числе Панксатонский Фил из города Панксатони, штат Пенсильвания. Кстати, именно он упоминается в фильме Билла Мюррея «День сурка», снятом в 1993 году.

Возможно, эта легенда обусловлена тем, что сурок — один из немногих видов животных, впадающих в настоящую зимнюю спячку. Будучи преимущественно растительноядными, летом сурки откармливаются растительностью, ягодами, орехами, насекомыми и овощами, которые выращивают люди, из-за чего многие считают их вредителями. Затем они выкапывают нору и в октябре впадают в спячку до марта (хотя могут проснуться раньше в районах с умеренным климатом или если оказываются в центре внимания в праздник «День сурка»).

Сурок — самый большой представитель семейства белок, его размер достигает 40–65 сантиметров, а вес — 2–4 килограммов. Имеет толстые изогнутые когти, идеально подходящие для копания земли, два слоя шерсти: плотный серый подшерсток и более светлый и длинный верхний слой, который защищает животное от непогоды.

Многие животные, изображенные на обложках книг издательства O'Reilly, находятся под угрозой вымирания; все они очень важны для биосферы. Чтобы узнать, чем вы можете помочь, посетите сайт animals.oreilly.com.

Мэтью Рассел, Михаил Классен

Data Mining. Извлечение информации из Facebook, Twitter, LinkedIn, Instagram, GitHub

Перевел с английского *А. Киселев*

Заведующая редакцией	<i>Ю. Сергиенко</i>
Ведущий редактор	<i>К. Тульцева</i>
Литературный редактор	<i>А. Бульченко</i>
Художественный редактор	<i>В. Мостипан</i>
Корректоры	<i>С. Беляева, М. Малчанова</i>
Верстка	<i>Л. Егорова</i>

Изготовлено в России Изготовитель: ООО «Прогресс книга».

Место нахождения и фактический адрес 194044, Россия, г. Санкт-Петербург,
Б Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления 08.2019 Наименование книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —

Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 23.07.19 Формат 70×100/16 Бумага офсетная. Усл. п. л. 37,410. Тираж 1200. Заказ 6113.

Отпечатано в АО «Первая Образцовая типография»

Филиал «Чеховский Печатный Двор»

142300, Московская область, г. Чехов, ул. Полиграфистов, д. 1

Сайт: www.chpd.ru, E-mail: sales@chpd.ru, тел. 8(499)270-73-59

ИЗДАТЕЛЬСКИЙ ДОМ «ПИТЕР» предлагает профессиональную, популярную и детскую развивающую литературу

Заказать книги оптом можно в наших представительствах

РОССИЯ

Санкт-Петербург: м. «Выборгская», Б. Сампсониевский пр., д. 29а
тел./факс: (812) 703-73-83, 703-73-72; e-mail: sales@piter.com

Москва: м. «Электrozаводская», Семеновская наб., д. 2/1, стр. 1, 6 этаж
тел./факс: (495) 234-38-15; e-mail: sales@msk.piter.com

Воронеж: тел.: 8 951 861-72-70; e-mail: hitsenko@piter.com

Екатеринбург: ул. Толедова, д. 43а; тел./факс: (343) 378-98-41, 378-98-42;
e-mail: office@ekat.piter.com; skype: ekat.manager2

Нижний Новгород: тел.: 8 930 712-75-13; e-mail: yashny@yandex.ru; skype: yashny1

Ростов-на-Дону: ул. Ульяновская, д. 26
тел./факс: (863) 269-91-22, 269-91-30; e-mail: piter-ug@rostov.piter.com

Самара: ул. Молодогвардейская, д. 33а, офис 223
тел./факс: (846) 277-89-79, 277-89-66; e-mail: pitvolga@mail.ru,
pitvolga@samara-ttk.ru

БЕЛАРУСЬ

Минск: ул. Розы Люксембург, д. 163; тел./факс: +37 517 208-80-01, 208-81-25;
e-mail: og@minsk.piter.com

Издательский дом «Питер» приглашает к сотрудничеству авторов:
тел./факс: (812) 703-73-72, (495) 234-38-15; e-mail: ivanova@piter.com
Подробная информация здесь: <http://www.piter.com/page/avtoru>

Издательский дом «Питер» приглашает к сотрудничеству зарубежных торговых партнеров или посредников, имеющих выход на зарубежный рынок: тел./факс: (812) 703-73-73; e-mail: sales@piter.com

Заказ книг для вузов и библиотек:
тел./факс: (812) 703-73-73, гоб. 6243; e-mail: uchebnik@piter.com

Заказ книг по почте: на сайте www.piter.com; тел.: (812) 703-73-74, гоб. 6216;
e-mail: books@piter.com

Вопросы по продаже электронных книг: тел.: (812) 703-73-74, гоб. 6217;
e-mail: kuznetsov@piter.com

Data Mining

ИЗВЛЕЧЕНИЕ ИНФОРМАЦИИ ИЗ СОЦИАЛЬНЫХ СЕТЕЙ

В недрах популярных социальных сетей — Twitter, Facebook, LinkedIn и Instagram — скрыты богатейшие залежи информации. Из этой книги исследователи, аналитики и разработчики узнают, как извлекать эти уникальные данные, используя код на Python, Jupyter Notebook или контейнеры Docker.

Сначала вы познакомитесь с функционалом самых популярных социальных сетей (Twitter, Facebook, LinkedIn, Instagram), веб-страниц, блогов и лент, электронной почты и GitHub. Затем приступите к анализу данных на примере Twitter.

Прочитайте эту книгу, чтобы

- Узнать о современном ландшафте социальных сетей
- Научиться использовать Docker, чтобы легко оперировать кодами, приведенными в книге
- Узнать, как адаптировать и поставлять код в открытый репозиторий GitHub
- Научиться анализировать собираемые данные с использованием возможностей Python 3
- Освоить продвинутые приемы анализа, такие как TFIDF, косинусное сходство, анализ словосочетаний, определение клика и распознавание образов
- Узнать, как создавать красивые визуализации данных с помощью Python и JavaScript

Мэтью Рассел — директор Built Technologies, он возглавляет команду лидеров, работающую над улучшением нашего мира. Вне работы Мэтью — рациональный индивидуалист, готовящийся к возможному зомби-апокалипсису.

Михаил Классен — главный специалист по обработке и анализу данных в Paladin AI, стартапе, занимающийся адаптивными технологиями обучения. Он увлекается проблемами искусственного интеллекта и анализом данных. Когда Михаил не занят на работе, он любит читать и путешествовать.

«Развитие когнитивных технологий расширяет наши возможности принятия решений на основе данных, поэтому книгу обязательно должны прочитать все, кто хочет получить практические навыки использования современных систем анализа данных, извлекаемых из самых ярких источников — социальных сетей».

Рохан Хаирнар, аналитик,
специализирующийся
на передовых приемах data mining,
Университет Шарлотт

«Умение получать полезную информацию из социальных сетей — жизненно необходимый навык в цифровую эпоху. Эта книга служит простым и исчерпывающим руководством для всех, кто желает использовать социальные данные с помощью самых современных методов их получения и анализа».

Стути Парамешвара Рао, аналитик,
магистр информационных технологий,
Университет Шарлотт



Заказ книг:

тел.: (812) 703-73-74
books@piter.com



instagram.com/piterbooks



youtube.com/ThePiterBooks



vk.com/piterbooks



facebook.com/piterbooks

WWW.PITER.COM

каталог книг и интернет-магазин

ISBN: 978-5-4461-1246-3



9 785446 112463