

O'REILLY®

Создание приложений машинного обучения

от идеи к продукту



Эммануэль
Амейзен



Building Machine Learning Powered Applications

Going from Idea to Product

Emmanuel Ameisen

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Создание приложений машинного обучения

от идеи к продукту

Эммануэль Амейзен



Санкт-Петербург • Москва • Минск

2023

ББК 32.973.2-018 + 32.813
УДК 004.41+004.85
А61

Амейзен Эммануэль

- А61 Создание приложений машинного обучения: от идеи к продукту. — СПб.: Питер, 2023. — 256 с.: ил. — (Серия «Библиотека программиста»).
- ISBN 978-5-4461-1773-4

Освойте ключевые навыки проектирования, разработки и развертывания приложений на базе машинного обучения (МО)!

Пошаговое руководство по созданию МО-приложений с упором на практику: для специалистов по обработке данных, разработчиков программного обеспечения и продакт-менеджеров.

Читая эту книгу, вы шаг за шагом создадите реальное практическое приложение — от идеи до внедрения. В вашем распоряжении примеры кодов, иллюстрации, скриншоты и интервью с ведущими специалистами в отрасли.

Вы научитесь планировать и измерять успех МО-проектов, разберетесь, как построить рабочую модель, освоите способы ее итеративной доработки. И, наконец, познакомитесь со стратегиями развертывания и мониторинга.

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

ББК 32.973.2-018 + 32.813
УДК 004.41+004.85

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 978-1492045113 англ.

Authorized Russian translation of the English edition of Building Machine Learning Powered Applications

ISBN 9781492045113 © 2020 Emmanuel Ameisen

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 978-5-4461-1773-4

© Перевод на русский язык ООО «Прогресс книга», 2022

© Издание на русском языке, оформление ООО «Прогресс книга», 2022

© Серия «Библиотека программиста», 2022

Оглавление

Предисловие	10
Зачем нужны приложения на базе машинного обучения	10
Используйте МО для создания практических приложений	10
Дополнительные ресурсы	11
МО на практике	12
Что вы найдете в книге	13
Необходимая подготовка	14
Наш учебный пример: написание текстов с использованием МО	14
Процесс МО как он есть	15
Условные обозначения	16
Использование исходного кода примеров	17
Благодарности	18
От издательства	18

ЧАСТЬ I

Выбор правильного подхода к машинному обучению

Глава 1. От цели продукта к разработке модели МО	21
Оценка осуществимости модели	22
Модели	24
Данные	31
Формулировка задачи по созданию редактора на основе МО	35
Делаем все с помощью МО: подход «от начала до конца»	35
Простейший подход: «сам себе алгоритм»	37
Золотая середина: обучение на полученном опыте	38
Моника Рогати: как выбирать и приоритизировать МО-проекты	40
Заключение	43
Глава 2. Составление плана	44
Оценка успешности	44
Производительность с точки зрения бизнеса	45

Производительность модели	46
Актуальность данных и сдвиг распределения	50
Скорость	52
Оценка масштаба и возможных проблем	53
Накапливайте знания в предметной области	53
Используйте опыт предшественников	55
Составление плана разработки МО-редактора	59
Начальный план разработки редактора	59
Всегда начинайте с простой модели	60
Как обеспечить устойчивый прогресс? Начинайте с простейшего решения	61
Начинайте с простого пайплайна	61
Пайплайн для МО-редактора	63
Заключение	65

ЧАСТЬ II

Создание рабочего пайплайна

Глава 3. Создание первого сквозного пайплайна	68
Простейший «каркас» приложения	68
Прототип МО-редактора	70
Парсинг и очистка данных	70
Токенизация текста	71
Генерирование признаков	72
Оцените рабочий процесс	74
Пользовательский опыт	74
Результаты моделирования	75
Оценка прототипа МО-редактора	76
Модель	77
Пользовательский опыт	78
Заключение	78
Глава 4. Получение исходного датасета	80
Итеративная доработка датасета	80
Проведите анализ данных	81
Изучение первого датасета	82
Начните с малого	82
Инсайт vs продукты	83
Критерии оценки качества данных	84

Разметка для выявления трендов в данных.....	91
Сводная статистика	91
Исследуйте и размечайте эффективно.....	93
Станьте алгоритмом	110
Тренды в данных	112
Используйте данные для принятия решений о признаках и моделях	113
Создание признаков на основе закономерностей.....	113
Признаки МО-редактора	117
Роберт Манро: как находить, размечать и использовать данные	118
Заключение	120

ЧАСТЬ III

Итеративная доработка моделей

Глава 5. Обучение и оценка модели	123
Самая простая адекватная модель.....	123
Простая модель	124
От закономерностей к моделям	126
Разделение датасета	128
Разделение данных МО-редактора	135
Оценка производительности.....	137
Оценка модели: не ограничивайтесь точностью	140
Сравнение предсказаний с данными.....	140
Матрица ошибок.....	141
ROC-кривая.....	142
Калибровочная кривая	145
Снижение размерности для анализа ошибок.....	145
Метод первых k элементов	147
Другие модели	151
Оценка важности признаков	152
Непосредственная оценка классификатора.....	153
Интерпретаторы «черного ящика».....	154
Заключение	156
Глава 6. Отладка МО-приложения	157
Передовые практики разработки ПО	157
Передовые практики для МО-приложений	159
Отладка потока данных: визуализация и тестирование	160

Начните с одного образца	160
Тестирование МО-кода	167
Отладка обучения: заставьте модель учиться	172
Сложность задачи	173
Проблемы оптимизации.....	175
Отладка обобщения: модель должна быть полезной	178
Утечка данных	178
Переобучение	179
Изучение решаемой задачи	183
Заключение	183

Глава 7. Использование классификаторов для выдачи рекомендаций 184

Вывод рекомендаций	185
Чего мы можем добиться без модели?	185
Извлечение глобальной важности признаков	187
Использование балла модели	188
Извлечение локальной важности признаков	188
Сравнение моделей	191
Версия 1: простейший отчет.....	191
Версия 2: более мощная, менее понятная	192
Версия 3: понятные рекомендации.....	194
Создание рекомендаций по редактированию текста	195
Заключение	199

ЧАСТЬ IV

Развертывание и мониторинг

Глава 8. Что еще учесть при развертывании модели 203

Забота о данных	204
Право собственности на данные.....	204
Смещение данных	205
Систематическое смещение	207
Забота о модели	208
Циклы обратной связи	208
Инклюзивная производительность модели	210
О контексте	211
Мошенники	212
Риск злоупотребления и двойного назначения	213

Крис Харланд: опыт поставки продуктов	214
Заключение	217
Глава 9. Выбор варианта развертывания	218
Развертывание на сервере	218
Потоковое приложение или API	219
Пакетные предсказания	221
Развертывание на стороне клиента.....	223
Развертывание на устройстве	225
Развертывание в браузере.....	227
Федеративное обучение: комбинированный подход.....	227
Заключение	229
Глава 10. Создание защитных механизмов для моделей.....	231
Проектирование с учетом возможных сбоев	231
Проверка входных данных и результатов.....	232
Резервные варианты на случай сбоя модели.....	236
Проектирование для обеспечения высокой производительности	240
Масштабирование при возрастании числа пользователей	241
Управление жизненным циклом модели и данных.....	244
Обработка данных и DAG.....	247
Запрос обратной связи.....	249
Крис Муди: специалисты по данным отвечают за весь пайплайн моделирования	252
Заключение	254
Глава 11. Мониторинг и обновление моделей.....	255
Мониторинг спасает жизни	255
Мониторинг для определения частоты обновлений	256
Мониторьте, чтобы выявить злоупотребления.....	256
Что мониторить	257
Метрики производительности	258
Бизнес-метрики	260
CI/CD для МО	261
A/B-тестирование и эксперименты	263
Другие подходы.....	266
Заключение	268
Об авторе	270
Иллюстрация на обложке.....	271

Предисловие

Зачем нужны приложения на базе машинного обучения

За последнее десятилетие на базе машинного обучения (МО) появляется все больше таких продуктов, как автоматизированные системы поддержки, сервисы перевода, рекомендательные системы, модели обнаружения мошенничества и т. д.

Что удивительно, информационных ресурсов по этой теме для инженеров и исследователей пока что не так много. Есть множество пособий и курсов по обучению МО-моделей и отдельно по разработке программного обеспечения. Но ресурсы, сводящие эти две сферы воедино для создания практических приложений на базе МО, можно пересчитать по пальцам.

Развертывание МО как составной части приложения требует креативности, аналитического склада ума и применения эффективных инженерных практик. Разрабатывать продукты на базе МО чрезвычайно сложно, поскольку этот процесс отнюдь не ограничивается простым обучением модели на датасете. Каждая из базовых составляющих процесса разработки МО-продукта: выбор подходящего метода для реализуемой функции, анализ ошибок моделей и проблем качества данных, валидация результатов модели в целях гарантии качества продукта — это вызов.

Наша книга проведет вас по всем этапам этого процесса, от начала и до конца. В ней содержится множество разных методов, примеров кода и рекомендаций от меня и других опытных практиков. Вы узнаете, какие навыки необходимы для проектирования, создания и развертывания приложений на базе МО. Цель книги — помочь вам преуспеть во всех составляющих МО-процесса.

Используйте МО для создания практических приложений

Если вы время от времени читаете статьи и корпоративные инженерные блоги по МО, то знаете, насколько они перегружены пугающим сочетанием линейных алгебраических уравнений и инженерной терминологии. В силу своего смешан-

ного характера сфера МО часто отпугивает многих инженеров и ученых, которые могли бы внести свой вклад в эту отрасль благодаря накопленному опыту. Примерно так же предприниматели и продакт-менеджеры испытывают трудности в соотнесении своих бизнес-идей с теми возможностями, которые сегодня предоставляет МО (и возможностями, которые станут доступны в будущем).

Эта книга написана с учетом опыта, который я получил за время, пока возглавлял программу ИИ в компании Insight Data Science. Я работал с командами по обработке данных во многих компаниях, помогая специалистам по Data Science, разработчикам ПО и продакт-менеджерам создавать практические полезные приложения на базе МО.

Цель книги — дать читателю пошаговое руководство по созданию таких приложений с упором на практику. Акцент делается на конкретных рекомендациях и методах, которые пригодятся при разработке прототипов, итеративной разработке и развертывании моделей. Поскольку книга охватывает широкий круг вопросов, каждый этап освещается настолько подробно, насколько это требуется для понимания темы. По мере возможностей я стараюсь приводить ссылки на информационные ресурсы, где можно найти более детальные сведения по каждой теме.

Важные концепции демонстрируются на реальных примерах. Кроме того, я покажу вам учебный проект разработки приложения. На протяжении всей книги мы будем реализовывать этот проект от исходной идеи до развернутой модели. Большинство примеров сопровождается иллюстрациями, многие — программным кодом. Весь код, приведенный в этой книге, можно найти в GitHub-репозитории, который является дополнением к книге¹.

Поскольку книга посвящена описанию всего процесса разработки приложений на базе МО, каждая новая глава опирается на концепции из предшествующих глав. Потому я рекомендую вам читать книгу последовательно. Так будет легче понять, как каждый этап встраивается в общий процесс. Если вам нужно детально изучить лишь некоторые части процесса разработки на базе МО, то, возможно, лучше обратиться к более специализированным книгам. На этот случай я приведу ниже ссылки на некоторые дополнительные ресурсы.

Дополнительные ресурсы

- Если вы хотите изучить МО так, чтобы уметь писать с нуля собственные алгоритмы, я рекомендую вам книгу Джоэла Грасса (Joel Grus) «*Data Science from Scratch*»². Если же вас интересуют теоретические основы глубокого обучения, то исчерпывающим руководством по этой теме является учебное по-

¹ <https://github.com/hundredblocks/ml-powered-applications>

² Грасс Дж. Data Science. Наука о данных с нуля.

собие «*Deep Learning*» (MIT Press), авторы Ян Гудфеллоу (Ian Goodfellow), Йошуа Бенджио (Yoshua Bengio) и Аарон Курвиль (Aaron Courville)¹.

- Если вы задаетесь вопросом эффективного и точного обучения моделей на конкретных датасетах, то вам будет полезно посетить сайты сообщества Kaggle² и компании fast.ai³.
- Если вы хотите создавать масштабируемые приложения, способные обрабатывать большие объемы данных, я рекомендую обратиться к книге Мартина Клеппмана (Martin Kleppmann) «*Designing Data-Intensive Applications*» (O'Reilly)⁴.

Если у вас есть некоторый опыт программирования и общее представление о МО и вы хотите создавать МО-продукты, то моя книга проведет вас по всему процессу — от исходной идеи продукта до поставки прототипа. Если вы уже работаете специалистом по обработке данных или инженером МО, то книга пополнит арсенал инструментов разработки. Если вы не пишете код, но вам необходимо сотрудничать со специалистами по обработке данных, то книга внесет ясность в ваши представления о процессе разработки МО-продуктов (если вы готовы пропустить некоторые подробные примеры кода).

Давайте для начала разберемся, что такое МО на практике.

МО на практике

Мы можем рассмотреть МО как процесс эффективного использования закономерностей в данных для автоматической настройки алгоритмов. Такое общее определение объясняет, почему МО начинают интегрировать в базовые механизмы многих приложений, инструментов и сервисов.

С некоторыми из таких продуктов пользователи сталкиваются непосредственно, например с поисковыми системами, рекомендациями в социальных сетях, сервисами перевода, а также системами автоматического распознавания лиц на фотографиях, выполнения голосовых команд или выдачи подсказок при написании текста электронного письма.

Другие работают менее очевидным образом, незаметно блокируя спам и мошенников, показывая рекламу, предсказывая закономерности поведения для более эффективного распределения ресурсов или персонализируя для пользователя интерфейс веб-сайта.

¹ Бенджио И., Гудфеллоу Я., Курвиль А. Глубокое обучение.

² <https://www.kaggle.com/>

³ <https://fast.ai>

⁴ Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. — СПб.: Питер.

Практическое использование МО означает выявление тех задач, которые выиграют от применения МО, и их успешное решение. Пройти весь путь от общей идеи МО-продукта до его конечной реализации достаточно сложно, и в этом вам поможет книга.

Некоторые курсы позволяют научиться методам МО путем обучения моделей на некотором датасете, однако обучение алгоритма на наборе данных — это лишь небольшая составная часть общего процесса. Востребованный продукт на базе МО опирается не только на совокупную оценку точности, он представляет собой результат длительной работы. В этой книге будут последовательно рассмотрены все этапы разработки такого продукта, начиная с идеи и заканчивая выпуском в производство. Каждый этап будет демонстрироваться на примере приложения. Я познакомлю вас с инструментами, передовыми практиками и распространенными ловушками, встречавшимися в ходе сотрудничества с командами разработчиков, для которых развертывание таких систем — будничная рутина.

Что вы найдете в книге

Чтобы как следует осветить тему создания МО-приложений, я постарался сделать содержание книги как можно более конкретизированным и практическим, в частности за счет демонстрации всего процесса на примерах.

Поэтому при рассказе о каждом этапе процесса я сначала буду описывать соответствующие методы, а затем демонстрировать их применение с помощью учебного примера. Книга также содержит множество реальных примеров использования МО в отрасли и интервью с профессионалами, имеющими опыт создания и эксплуатации МО-моделей.

Полное описание процесса МО

Чтобы успешно поставить пользователям продукт на базе МО, недостаточно просто обучить модель. Вы должны надлежащим образом *транслировать* потребности продукта в задачу МО, *собрать* достаточный объем данных, эффективно выполнить *итеративную доработку* модели, *валидировать* результаты, а затем *развернуть* модель, обеспечив ее устойчивость к ошибкам.

Создание модели часто составляет лишь десятую часть от общего объема работ по реализации проекта МО. Надлежащее освоение всех этапов пайплайна МО позволит вам успешно создавать проекты, легко проходить собеседования на позиции, связанные с машинным обучением, и вносить весомый вклад в работу команд МО.

Технически и практически ориентированный пример

Хотя книга не научит вас реализации алгоритмов МО с нуля на языке Си, она носит достаточно практический и технический характер, демонстрируя при-

менение библиотек и инструментов для более высокого уровня абстракции. По ходу книги мы с вами разберем пример создания МО-приложения, начиная с исходной идеи и заканчивая развертыванием продукта.

Там, где это уместно, я буду иллюстрировать ключевые понятия с помощью фрагментов кода и рисунков. Поскольку лучший способ освоить МО состоит в его практическом применении, я настоятельно рекомендую вам воспроизводить примеры из книги, адаптируя их для создания собственного приложения на базе МО.

Реальные проекты

По ходу книги я буду приводить содержание бесед и рекомендации от ведущих специалистов в области МО, которым довелось поработать в командах по обработке данных таких IT-компаний, как StitchFix, Jawbone и FigureEight. В материалах содержатся практические выводы, полученные из опыта создания МО-приложений с миллионами пользователей. Это избавит вас от некоторых типичных заблуждений относительно того, что приносит успех специалистам и командам в области обработки данных.

Необходимая подготовка

Книга написана с расчетом на то, что читатель имеет некоторое представление о программировании. В большинстве технических примеров я буду использовать язык Python, предполагая, что читатель знаком с его синтаксисом. Если вы хотите освежить свои знания языка Python, я могу порекомендовать вам книгу Кеннета Рейтца (Kenneth Reitz) и Тани Шлюссер (Tanya Schlusser) «*The Hitchhiker's Guide to Python*» (O'Reilly)¹.

Также хочу заметить, что, хотя я буду всегда давать определение концепций МО, я не буду касаться внутреннего устройства алгоритмов МО. В большинстве случаев эти алгоритмы представляют собой стандартные методы МО, описание которых можно найти в пособиях из раздела «Дополнительные ресурсы» на с. 11.

Наш учебный пример: написание текстов с использованием МО

Для иллюстрации идей на конкретном примере по ходу книги мы вместе с вами будем разрабатывать МО-приложение.

¹ Рейтц К., Шлюссер Т. Автостопом по Python. — СПб.: Питер.

В качестве учебного примера я выбрал приложение, хорошо отражающее всю сложность итеративной доработки и развертывания МО-моделей. Я также хотел, чтобы это был полезный продукт. Итак, мы будем создавать *помощника по написанию текстов*.

Задача — разработать систему, которая помогает пользователям писать более качественные тексты или вопросы. Может показаться, что эта цель носит достаточно расплывчатый характер, но позже я дам ей более четкое определение. Учебный проект обладает рядом важных достоинств:

Текстовые данные используются повсеместно

Текстовые данные есть почти везде, где вы только можете вообразить, и они играют ключевую роль во многих практических МО-приложениях. Когда мы пытаемся лучше разобраться в отзывах о нашем продукте, точнее классифицировать запросы, поступающие в службу поддержки, или скорректировать свои рекламные сообщения с учетом целевой аудитории, мы принимаем и производим некоторые текстовые данные.

Помощники по написанию текстов приносят реальную пользу

Как показывает практика, МО-редакторы, начиная с функции умного ввода текста в почтовом сервисе Gmail и заканчивая продвинутой проверкой орфографии сервиса Grammarly, могут предоставлять пользователям множество разнообразных возможностей. Вам будет интересно узнать, как создать такое приложение с нуля.

Помощник на базе МО — самостоятельное приложение

Многие МО-приложения могут работать только при интеграции в более широкую экосистему. Например, предсказания времени ожидания в сервисах заказа такси, поиск и выдача рекомендаций в интернет-магазинах, модели для проведения рекламных аукционов. В отличие от таких приложений текстовый редактор, хотя и может получить дополнительную пользу от интеграции в экосистему редактирования документов, часто является полезным сам по себе и может быть предоставлен пользователю через простой веб-сайт.

На протяжении всей книги учебный проект поможет нам в иллюстрации сложных задач, возникающих в процессе создания МО-приложений, и предлагаемых методов их решения.

Процесс МО как он есть

Путь от исходной идеи до развернутого МО-приложения извилист и тернист. Проанализировав опыт многих компаний и специалистов, я выделил четыре основных последовательных этапа, каждому из которых будет посвящена отдельная часть книги.

1. *Определение правильного подхода к МО.* Машинное обучение — достаточно широкая сфера и часто предлагает много способов достижения цели, поставленной при создании продукта. Выбор наилучшего подхода в каждом конкретном случае будет зависеть от многих факторов, таких как критерии успешности, доступность данных и степень сложности задачи. На данном этапе необходимо правильно определить критерии успешности, достаточный размер исходного датасета и подходящий тип модели.
2. *Создание исходного прототипа.* Перед тем как начать работу над моделью, необходимо создать «сквозной» прототип, учитывающий все факторы. Такой прототип должен достигать целей, для которых создавался продукт, без МО. Создание прототипа обычно позволяет понять, уместно ли использование МО, и дает возможность начать сбор данных для обучения модели.
3. *Итеративная доработка моделей.* Собрав датасет, вы можете приступить к обучению модели и оценке ее недостатков. Этот этап сводится к циклу анализа ошибок и корректировки реализации. Ускорение этого цикла является самым эффективным способом ускорения процесса МО-разработки.
4. *Развертывание и мониторинг.* Добившись хорошей производительности модели, вы должны выбрать подходящий вариант развертывания. После развертывания модели часто происходит ее непредвиденный отказ. Последние две главы этой книги будут посвящены методам предотвращения и мониторинга ошибок.

Как видите, нам предстоит поистине длинный путь, давайте сейчас же начнем его с главы 1.

Условные обозначения

В этой книге используются следующие условные обозначения.

Курсив

Курсивом выделены новые термины или важные понятия.

Моноширинный шрифт

Используется для листингов программ, а также внутри абзацев для обозначения таких элементов, как переменные и функции, базы данных, типы данных, переменные среды, операторы и ключевые слова, имена файлов и их расширения.

Моноширинный полужирный шрифт

Показывает команды или другой текст, который пользователь должен ввести самостоятельно.

Моноширинный курсив

Показывает текст, который должен быть заменен значениями, введенными пользователем, или значениями, определяемыми контекстом.

Шрифт без засечек

Используется для обозначения URL, адресов электронной почты, названий кнопок и других элементов интерфейса, каталогов.



Этот рисунок указывает на совет или предложение.



Этот рисунок указывает на общее примечание.



Этот рисунок указывает на предупреждение.

Использование исходного кода примеров

Вспомогательные материалы (примеры кода, упражнения и т. д.) доступны для загрузки по адресу: <https://github.com/hundredblocks/ml-powered-applications>. Если у вас возникнут вопросы технического характера по использованию примеров кода, направляйте их по электронной почте на адрес bookquestions@oreilly.com.

В общем случае все примеры кода из 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. Если вы разрабатываете программу и используете в ней несколько фрагментов кода из книги, вам не нужно обращаться за разрешением. Но для продажи или распространения примеров из книги вам потребуется разрешение от издательства O'Reilly. Вы можете отвечать на вопросы, цитируя данную книгу или примеры из нее, но для включения существенных объемов программного кода из книги в документацию вашего продукта потребуется разрешение.

Мы рекомендуем, но не требуем добавлять ссылку на первоисточник при цитировании. Под ссылкой на первоисточник мы подразумеваем указание авторов, издательства и ISBN.

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

Благодарности

Проект этой книги зародился как результат моего менторства и курирования МО-проектов в компании Insight Data Science. Я хотел бы поблагодарить Джейка Кламку (Jake Klamka) за то, что дал мне возможность возглавить эту программу, и Джереми Карновски (Jeremy Karnowski) за то, что подтолкнул меня к тому, чтобы изложить на бумаге опыт, полученный в ходе этой работы. Я также хотел бы поблагодарить сотни участников менторской программы, с которыми мне довелось поработать в компании Insight Data Science, за то, что помогли расширить границы представления об МО-проектах.

Написание книги — пугающе сложная задача, в решении которой мне очень помогла постоянная поддержка сотрудников компании O'Reilly. В частности, хочу сказать спасибо моему редактору Мелиссе Поттер (Melissa Potter) за ее полезные советы и моральную поддержку на протяжении всего процесса. Спасибо Майку Лоукидесу (Mike Loukides) за то, что каким-то образом убедил меня в том, что написание книги — вполне разумное желание.

Спасибо научным редакторам, которые тщательно проверяли первые черновики этой книги, находили ошибки и давали советы по улучшению ее содержания. Спасибо вам, Алекс Гуде (Alex Gude), Джон Крон (Jon Krohn), Кристен Макинтайр (Kristen McIntyre) и Дуве Осинга (Douwe Osinga), за то, что нашли время в своем плотном графике и помогли сделать эту книгу лучше. Я также очень признателен практикующим специалистам по обработке данных, рассказавших о сложностях МО, с которыми они столкнулись на практике и посчитали достойными внимания. Спасибо вам за то, что уделите мне время и предложили свои идеи, надеюсь, что я достаточно подробно изложил их в этой книге.

Наконец, за неустанную поддержку на протяжении долгих вечеров и выходных дней, в которые я писал эту книгу, хочу поблагодарить свою верную подругу жизни Мари, моего язвительного помощника Эллиота, мою мудрую и терпеливую семью, а также своих друзей, которые прикрывали меня во время моего отсутствия на работе. Эта книга появилась лишь благодаря вашей поддержке.

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

Выбор правильного подхода к машинному обучению

Большинство разработчиков и компаний хорошо понимают, как решать свои задачи: например, как спрогнозировать уход клиентов с онлайн-платформы или как создать дрон, который будет следить за лыжником, спускающимся с горы. И точно так же большинство людей может быстро разобраться, как обучать модель, которая будет с приемлемой точностью классифицировать потребителей или искать объекты на основе датасета.

Однако очень редко разработчики способны взять задачу, определить наилучший способ ее решения, выбрать алгоритм машинного обучения (МО) и уверенно реализовать его. Часто это умение приходит лишь с опытом, после множества сверхамбициозных проектов и сорванных сроков выполнения.

Для каждого проекта есть несколько способов реализации с помощью МО. Слева на рисунке (рис.. I.1) представлен макет помощника по написанию текстов. Помощник дает рекомендации и получает от пользователя обратную связь. Справа на рисунке — схема метода МО для выдачи этих рекомендаций.

В начале этого раздела мы рассмотрим подходы к МО и определим наиболее предпочтительные. Затем разберем, как согласовать метрики производительности модели с требованиями к продукту.



Рис. I.1. От продукта к МО

Раздел состоит из двух глав.

Глава 1

К концу этой главы вы сможете проанализировать идею для приложения, оценить, реально ли ее осуществить и понадобится ли для этого МО, а затем выбрать начальную модель.

Глава 2

В этой главе мы расскажем, как наиболее точно оценить производительность модели в зависимости от целей вашего приложения и как использовать этот показатель для улучшения работы модели.

От цели продукта к разработке модели МО

В технологиях МО машина учится на основе данных. Задается цель, и с помощью вероятностных методов машина решает задачу, постепенно приближаясь к этой цели. Такой подход коренным образом отличается от традиционного программирования, когда разработчик пишет пошаговые инструкции, указывающие, как следует решать задачу. В силу этого МО хорошо подходит для задач, где невозможно определить эвристическое решение.

На рис. 1.1 показаны два способа создания системы распознавания кошек. Слева — традиционная программа, для которой процедуры написаны вручную. Справа — модель, обученная на размеченных фотографиях кошек и собак. Такой метод МО позволяет относить изображения к правильной категории, при этом не нужно указывать, как получить необходимый результат, достаточно лишь примеров входных и выходных данных.



Рис. 1.1. От ручного написания процедур к прямой подаче данных

МО — мощный инструмент, позволяющий создавать совершенно новые виды продуктов. Однако МО опирается на распознавание образов, а это вносит некоторую неопределенность. Поэтому важно выяснить, какие элементы продукта

выиграют от применения МО, и сформулировать цели обучения так, чтобы свести к минимуму вероятность разочарования пользователей.

Так, например, едва ли человек способен написать пошаговые инструкции для автоматического распознавания животных по значению отдельных пикселей изображения (а если и способен, то потратит на это немало времени). Однако, «скормив» тысячи изображений различных животных сверточной нейронной сети (СНС), можно создать модель, которая будет справляться с этой задачей даже лучше человека. Для решения такой задачи будет уместно использовать МО.

Применение МО для автоматического расчета налогов, напротив, будет неразумным решением, поскольку такие системы должны работать в соответствии с нормами налогового законодательства и, как многие знают, ошибки в налоговой декларации недопустимы.

МО не следует использовать в случаях, когда задачу можно решить с помощью набора детерминированных правил, которые несложно написать и поддерживать.

Итак, МО открывает новый мир самых разных приложений, однако важно понимать, какие задачи *можно* и *нужно* решать с его помощью. При создании продуктов необходимо прежде всего определить, требуется ли МО для конкретной бизнес-задачи, а затем подобрать метод, позволяющий максимально быстро выполнять итеративную доработку.

В этой главе мы расскажем, как определить, требуется ли МО для решения задачи, а также рассмотрим методы МО и требования при работе с данными в зависимости от цели продукта. Я продемонстрирую эти приемы на учебном примере — редакторе на базе МО, о котором мы уже говорили в разделе «Наш учебный пример: написание текстов с использованием МО» на с. 14. В конце главы вас ждет интервью с Моникой Рогати (Monica Rogati).

Оценка осуществимости модели

Исходя из того, что модели на базе МО способны работать без пошаговых инструкций, они могут выполнять определенные задачи лучше экспертов, например, выявлять опухоли на рентгеновских снимках или играть в го. Они также могут решать совершенно невозможные для человека задачи, например, среди миллиона статей выбирать и рекомендовать подходящую или изменять голос говорящего, чтобы он звучал так, как будто говорит другой человек.

Способность систем МО обучаться непосредственно на данных означает, что МО возможно использовать в широком спектре приложений. Однако человеку становится все сложнее разобраться в том, какие задачи можно реально решить

с помощью МО. На каждую успешно решенную задачу, упомянутую в научной статье или публикации в корпоративном блоге, приходится сотни неудачных решений.

Пока что нет безошибочных способов предсказать, насколько успешным будет применение МО, однако есть ряд рекомендаций для снижения рисков. Самая главная рекомендация — всегда отталкиваться от цели продукта и находить наилучший способ ее достижения.

На этом этапе следует быть открытым для любых подходов безотносительно того, требуют ли они применения МО. При рассмотрении подходов к МО мало оценки того, насколько они интересны сами по себе. Обязательно оцените и то, насколько они подходят для вашего продукта.

Лучше всего уложиться в два последовательных этапа: (1) сформулируйте цель вашего продукта в парадигме МО; (2) оцените осуществимость этой задачи для МО. В зависимости от полученной оценки можно корректировать процесс разработки до тех пор, пока вас не устроит результат. Давайте посмотрим, что в действительности включают в себя эти два этапа.

1. *Сформулировать цель продукта в парадигме МО.* При создании продукта в первую очередь следует подумать о том, какой сервис мы хотим предоставить пользователям. Как уже упоминалось в предисловии, в этой книге концепции будут иллюстрироваться на учебном примере — редакторе, который помогает пользователям грамотнее писать сообщения с вопросами. Цель такого продукта понятна: давать пользователям практические и полезные рекомендации по написанию текста. Однако задачи МО формулируются иначе: *обучение модели и приобретение функционала на основе данных*. Наглядный пример такого обучения — преобразование текста на одном языке в текст на другом. Для одной цели продукта обычно можно по-разному сформулировать задачу, сложность которой тоже может варьироваться.
2. *Оценить осуществимость МО.* Задачи МО могут сильно различаться. По мере развития МО задачи вроде классификации фотографий кошек и собак решаются буквально за пару часов, в то время как, например, создание системы, способной вести полноценный разговор, пока остается нерешенной научной проблемой. Для эффективной разработки МО-приложений важно рассмотреть несколько формулировок задачи и взять за основу самую простую. Лучший способ оценить сложность МО — обдумать, какие данные ей нужны и какие из ныне существующих моделей способны работать с такими данными.

Прежде чем рассмотреть несколько формулировок и оценить их осуществимость, нам следует изучить две базовые составляющие задач МО — данные и модели.

Начнем с моделей.

Модели

Существует много широко используемых моделей МО, мы не станем рассматривать здесь каждую из них. Их подробное описание вы можете найти в книгах из раздела «Дополнительные ресурсы» на с. 11. Кроме известных моделей, еженедельно появляются новые вариации существующих, новые архитектуры и техники оптимизации. Только за май 2019 года ArXiv¹, популярный электронный архив научных статей, где часто появляются статьи о новых моделях, пополнился 13 тысячами новых публикаций.

В то же время полезно иметь обзор различных категорий моделей и разобраться, к каким задачам они применимы. Поэтому я составлю простейшую таксономию моделей, основываясь на их подходе к задаче. Вы можете руководствоваться этой таксономией при выборе подхода к решению конкретной задачи МО. Поскольку при машинном обучении модели и данные тесно связаны друг с другом, содержание этого раздела во многом пересекается с содержанием раздела «Типы данных» (с. 32).

Алгоритмы МО можно классифицировать, основываясь на том, нужны ли метки для обучения. Под метками здесь имеется в виду присутствие в данных целевого параметра, который мы должны получать от обученной модели. Алгоритмы обучения с учителем как раз оперируют с такими наборами данных. Алгоритмы обучения без учителя, напротив, не требуют меток. Наконец, алгоритмы с частичным привлечением учителя используют метки, которые не являются желаемыми выходными данными, но в некотором роде с ними схожи.

Во многих случаях цель продукта можно реализовать с помощью алгоритмов обучения как с учителем, так и без него. Можно создать систему обнаружения мошенничества, научив модель выявлять те транзакции, которые чем-то отличаются от типичных, что не потребует использования меток. В то же время такую систему можно создать, вручную пометив мошеннические и законные транзакции и обучив модель с этими метками.

В большинстве случаев алгоритмы обучения с учителем легче валидировать, поскольку можно использовать метки для оценки качества предсказаний, полученных от модели. Доступ к целевым параметрам также облегчает процесс обучения модели. Хотя создание датасета с метками часто требует больших затрат времени на этапе предварительной подготовки, это сильно упрощает создание и валидацию моделей. Поэтому в этой книге мы будем рассматривать главным образом обучение с учителем.

В то же время, выяснив, какие входные данные будет принимать ваша модель и какие данные она будет возвращать, вы существенно сократите количество

¹ <https://arxiv.org>

возможных подходов к моделированию. В зависимости от этих данных могут подойти следующие типы задач МО:

- классификация и регрессия;
- извлечение знаний;
- каталогизация;
- генеративные модели.

Мы подробно рассмотрим каждую из этих категорий в последующих разделах. При изучении разных подходов к моделированию я рекомендую вам учитывать то, какие данные у вас уже есть или какие вы можете собрать. Именно доступность данных часто становится ограничивающим фактором при выборе модели.

Классификация и регрессия

В некоторых проектах ставится цель эффективно классифицировать данные по двум или более категориям или присвоить им значения по непрерывной шкале (что называется уже не «классификацией», а «регрессией»). Хотя технически регрессия и классификация — это разные задачи, методы их реализации имеют между собой много общего, что позволяет нам объединить их в одну категорию.

Сходство классификации и регрессии обусловлено тем, что большинство моделей для классификации выдает в качестве результата вероятность принадлежности элемента данных к той или иной категории. Выходит так, что суть классификации состоит в том, чтобы каким-то образом отнести объект к той или иной категории на основе заданных показателей. Если не вдаваться в подробности, вероятностные показатели классификации можно рассматривать как регрессию.

Обычно нам приходится классифицировать или оценивать отдельные образцы, как это делают спам-фильтры, сортирующие электронные письма на ценные и мусор, системы обнаружения мошенничества, классифицирующие пользователей на законопослушных и злоумышленников, или системы машинного анализа рентгеновских снимков, определяющие сломанные и здоровые кости.

На рис. 1.2 представлен пример классификации предложения на основе его эмоциональной окраски и темы.

В проектах на основе регрессии требуется не отнесение каждого образца к определенной категории, а присвоение ему определенного значения. Примером регрессионной задачи может служить предсказание цены продажи дома на основе таких характеристик, как количество комнат и местоположение.

В некоторых случаях для предсказания некоторого события в будущем можно использовать целый ряд изменений показателя за временной промежуток (вместо показаний в определенный момент). Такие данные часто называют *времен-*

ными рядами, а генерирование предсказаний на их основе — *прогнозированием*. Временные ряды данных могут быть представлены в виде истории болезни пациента или ряда оценок посещаемости национальных парков. В таких проектах часто выигрывают модели и признаки, которые могут использовать эти дополнительные временные характеристики.

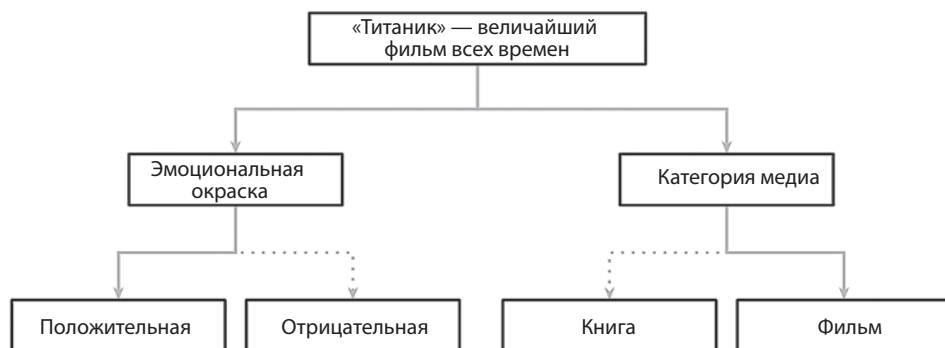


Рис. 1.2. Классификация предложения в рамках нескольких категорий

В некоторых случаях необходимо выявить в датасете необычные события, что называют *выявлением аномалий*. Когда задача классификации направлена на выявление аномальных случаев, которые составляют малую часть данных, часто требуется применять другой набор методов. Хорошей аналогией здесь будет поиск иголки в стоге сена.

Для качественной классификации и регрессии часто требуется провести значительную работу по отбору и генерации признаков. Отбор признаков подразумевает выделение подмножества признаков, имеющих наиболее высокую прогностическую ценность. Генерация признаков означает выявление и создание хороших предикторов целевого значения путем модификации или объединения имеющихся признаков датасета. Мы подробно рассмотрим оба этих процесса в части III.

В последнее время методы глубокого обучения показывают хорошие результаты в автоматическом извлечении полезных признаков из графических, текстовых данных и аудио. Вероятно, в будущем они упростят процесс создания и отбора признаков, но пока что эти методы остаются неотъемлемой частью общего процесса МО.

Итак, мы можем опираться на описанные выше методы классификации и количественной оценки для предоставления полезных рекомендаций. Для этого потребуются создать интерпретируемую модель классификации и использовать ее признаки для генерирования применимых на практике рекомендаций. О том, как это делается, будет подробно рассказано далее.

Надо сказать, что не все задачи нацелены на отнесение образца к одной из категорий или присвоение ему значения. Иногда требуется действовать более тонко, извлекая информацию из отдельных частей входных данных, например, чтобы определить, в какой части изображения расположен объект.

Извлечение знаний из неструктурированных данных

Структурированные данные — это данные, хранимые в табличном формате, например таблицы баз данных и электронные таблицы Excel. *Неструктурированные данные* — это датасеты, не представленные в табличном формате, например тексты (статьи, обзоры, содержимое Википедии и т. д.), музыкальные записи, видеозаписи или песни.

На рис. 1.3 показаны примеры структурированных и неструктурированных данных. Модели извлечения знаний берут источник неструктурированных данных и извлекают из него структуру, используя МО.

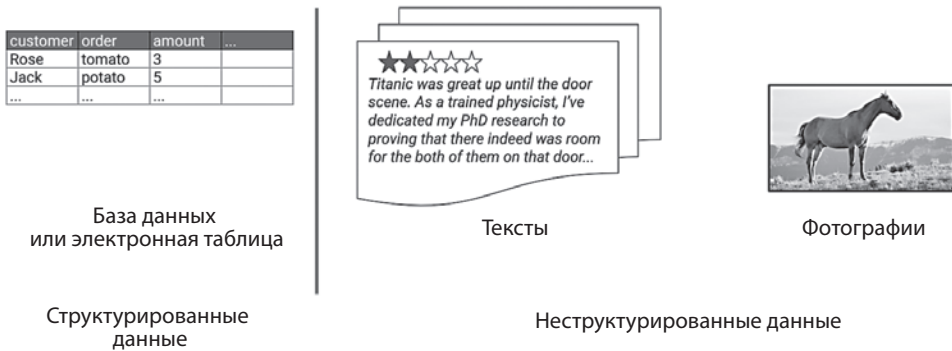


Рис. 1.3. Примеры структурированных и неструктурированных данных

Извлечение знаний из текста может использоваться, к примеру, для структурирования отзывов. Вы можете научить модель извлекать из отзывов сведения о таких характеристиках, как чистота, качество обслуживания или цена. Это позволит пользователям легко находить отзывы по интересующей их теме.

Для сферы медицины можно создать модель извлечения знаний, которая будет принимать в качестве входных данных медицинские документы и извлекать диагноз, историю и проявления болезни. В примере, показанном на рис. 1.4, модель принимает в качестве входных данных предложение и извлекает из него название и тип медиа. Применяя такую модель к комментариям на форуме поклонников кино, мы могли бы генерировать сводные отчеты о том, какие фильмы обсуждаются наиболее часто.



Рис. 1.4. Извлечение из предложения названия и типа медиа

Задача извлечения знаний может применяться для поиска на изображении интересующих областей и последующей их классификации по категориям. Два распространенных подхода показаны на рис. 1.5: более грубый метод обнаружения объектов подразумевает выделение интересующей области прямоугольником (или *ограничительной рамкой*), а метод *сегментации* производит точное отнесение пикселей изображения к рассматриваемой категории.



Рис. 1.5. Ограничительные рамки и маски сегментации

Иногда извлеченная информация может использоваться в качестве входных данных для другой модели. Так, мы могли бы извлекать ключевые точки из видеозаписи, демонстрирующей принятую йогом позу, с помощью модели для распознавания поз и «скармливать» эти ключевые точки второй модели, классифицирующей позу как правильную или неправильную на основе размеченных данных. Как может выглядеть такая цепочка из двух моделей, показано на рис. 1.6. При этом первая модель извлекает структурированную информацию (координаты суставов) из неструктурированных данных (фотографии), а вторая принимает эти координаты и классифицирует их как позу йоги.

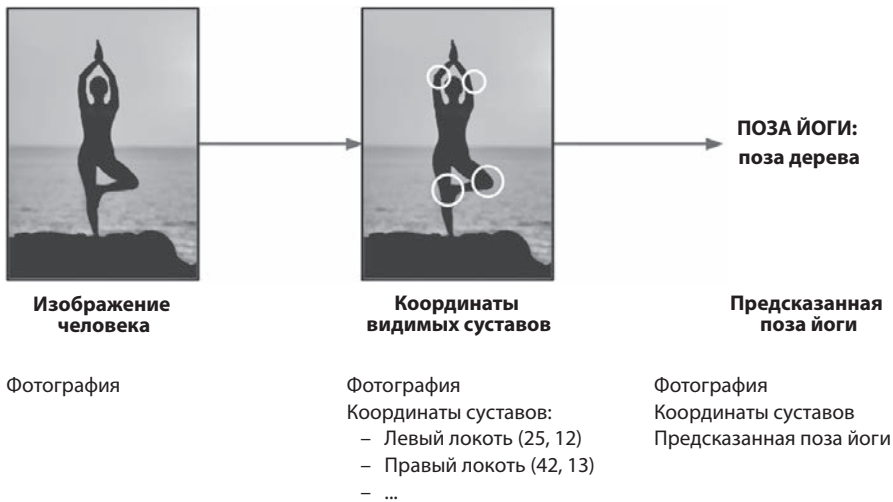


Рис. 1.6. Распознавание поз йоги

Модели, которые мы рассматривали до сих пор, генерируют новые данные на основе входных данных. Однако в некоторых случаях задача состоит в поиске релевантных объектов, например в случае поисковых и рекомендационных систем. Модели следующей категории служат как раз для этой цели.

Каталогизация

Модели каталогизации чаще всего генерируют для пользователя некий набор результатов. Эти результаты зависят от того, какую фразу ввел пользователь в строке поиска, какое изображение он загрузил на сервер или какую команду он дал голосовому помощнику. Часто набор результатов может быть представлен заранее, еще до ввода пользователем какого-либо запроса, например в виде рекомендаций к просмотру в стриминговых сервисах.

На рис. 1.7 показан пример системы, которая рекомендует пользователю фильмы, основываясь на том, какой фильм он только что просмотрел, — пользователю не нужно ничего искать.

Таким образом, эти модели либо *рекомендуют* объекты, имеющие некую связь с объектом, к которому пользователь уже проявил интерес (например, схожие статьи на сайте Medium или товары на сайте Amazon), или предлагают удобный способ *поиска* в каталоге (позволяя пользователям искать объекты путем ввода текста или загрузки своих фотографий).

Чаще всего такие рекомендательные системы обучаются на паттернах поведения пользователей в прошлом. В таком случае их называют *коллаборативными* рекомендательными системами. В случае, когда модели принимают отдельные



Рис. 1.7. Рекомендации фильмов к просмотру

признаки объектов, их называют *контент-ориентированными* рекомендательными системами. Некоторые системы одновременно используют и коллаборативный, и контент-ориентированный подход.

Наконец, МО также может использоваться для творчества. Вы можете научить модели генерировать эстетически приятные изображения или аудиозаписи или даже смешные анекдоты. Такие модели называют *генеративными*.

Генеративные модели

Генеративные модели генерируют данные, которые теоретически могут зависеть от пользовательского ввода. Такие модели не классифицируют, не делают точные расчеты, не извлекают и не организуют информацию, и поэтому у них обычно широкий диапазон возможных результатов на выходе. Это означает, что генеративные модели — уникальная находка для таких задач, как перевод с одного языка на другой, где результаты могут сильно варьироваться.

С другой стороны, из-за того, что генеративные модели часто выдают данные с широким разбросом, увеличиваются риски их применения на производстве. Поэтому, если такая модель не является обязательной для достижения вашей цели, я рекомендую вам сначала попробовать применить модели других типов. Тем читателям, которые хотели бы поглубже ознакомиться с темой генеративных моделей, я могу порекомендовать книгу Дэвида Фостера (David Foster) «Генеративное глубокое обучение» («Generative Deep Learning»¹).

¹ <https://learning.oreilly.com/library/view/generative-deep-learning/9781492041931/>

Примерами практического применения таких моделей могут служить системы перевода, которые конвертируют предложение на одном языке в предложение на другом; системы автоматического реферирования; системы генерирования субтитров, которые переводят аудио- и видеозаписи в текстовые расшифровки; системы нейронного переноса стиля (см. статью Гатиса (Gatys) и др. «Нейронный алгоритм для художественных стилей» («A Neural Algorithm of Artistic Style»¹)), которые преобразуют изображения в их стилизованные интерпретации.

На рис. 1.8 представлен пример генеративной модели, которая переносит на фотографию (слева) стиль живописи с миниатюры (справа, в нижнем левом углу).

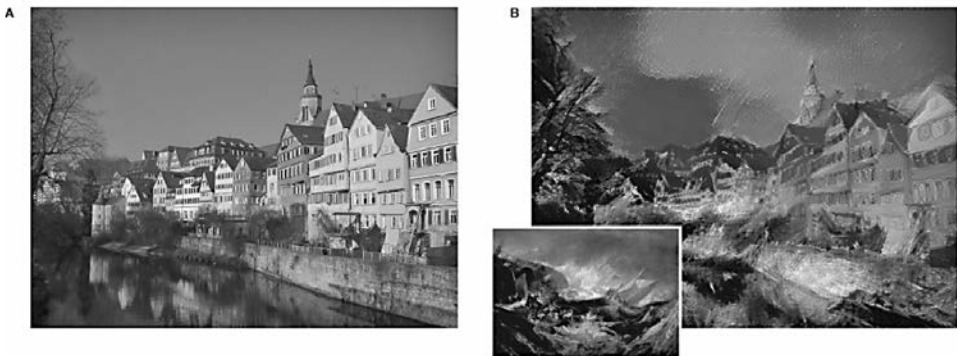


Рис. 1.8. Пример переноса стиля из статьи Гатиса (Gatys) и др. «Нейронный алгоритм для художественных стилей»

Как вы уже поняли, для обучения моделей разного типа требуется использовать разные типы данных. Обычно выбор модели в значительной мере зависит от того, какие данные вы можете получить, то есть выбор модели, как правило, определяется доступностью данных.

Теперь давайте рассмотрим несколько распространенных сценариев получения данных и соответствующие им модели.

Данные

Модели машинного обучения с учителем используют закономерности во входных данных так, чтобы на выходе получались наиболее полезные результаты. Если датасет содержит признаки, позволяющие предсказывать целевой результат, вы сможете обучить на нем модель. Однако в большинстве случаев

¹ <https://oreil.ly/XVwMs>

у нас изначально нет подходящих данных для обучения модели, реализующей юзкейс¹ от начала до конца.

Допустим, нам нужно обучить систему распознавания речи принимать голосовые команды пользователей, интерпретировать их и выполнять соответствующие действия. Начиная работу над этим проектом, мы можем определить набор команд для распознавания, например команду «включи фильм на телевизоре».

Чтобы обучить МО-модель выполнению этой задачи, нам потребуется датасет с образцами голосовых команд, произнесенных представителями различных социальных групп привычным для них языком. Наличие набора входных данных с необходимыми образцами играет критически важную роль, поскольку любая модель способна обучаться только на тех данных, которые мы ей предоставляем. Если датасет будет содержать команды, произнесенные представителями только одной группы населения, то и продукт будет полезен лишь для этой группы. Поскольку мы выбрали специализированную предметную область, крайне маловероятно, что датасет с подобными образцами уже существует.

При создании большинства приложений нам потребуется искать, проверять и собирать дополнительные данные. Процесс получения данных может сильно варьироваться по масштабу и сложности в зависимости от специфики проекта, поэтому чтобы продукт получился успешным, важно заранее оценить сложность стоящей перед вами задачи.

Для начала давайте разберемся в том, в какой исходной ситуации вы можете оказаться при поиске данных. Обычно она играет ключевую роль при выборе дальнейших действий.

Типы данных

Определив задачу как *сопоставление входных и выходных данных*, можно приступить к поиску источников данных.

В случае систем обнаружения мошенничества это могут быть данные о рядовых пользователях и злоумышленниках наряду с характерными чертами их аккаунтов, благодаря которым можно предсказывать поведение пользователей. В случае систем языковых переводов это может быть множество предложений на исходном языке и их перевод. В случае поисковых систем и агрегаторов контента это может быть история последних поисковых запросов и переходов по ссылкам.

Редко удастся точно определить схему получения нужного результата из входных данных, поэтому будет полезно рассмотреть несколько возможных случаев. Можно представить это как своего рода иерархию потребности в данных.

¹ Use case переводится как «сценарий использования», но в последнее время в профессиональной среде все чаще используется «юзкейс». — *Примеч. ред.*

Доступность данных

Грубо говоря, можно выделить три уровня доступности данных: от самого доступного к труднодоступному. К сожалению, как и в большинстве других задач, самые полезные данные собрать сложнее всего. Давайте перечислим типы доступности данных:

Доступность размеченных данных

Этот случай показан слева на рис. 1.9. Любой разработчик, реализующий обучение с учителем, мечтает о получении *размеченного датасета*. Под словом «размеченный» здесь понимается то, что многие элементы данных такого набора содержат целевое значение, которое должна предсказывать модель. Это существенно облегчает обучение и оценку качества модели, поскольку вы можете использовать метки в качестве эталонных ответов. На практике вероятность того, что вы найдете отвечающий вашим потребностям размеченный набор данных и он будет свободно доступен в интернете, обычно стремится к нулю. Нередко разработчики ошибочно принимают датасет, который они нашли, за нужный.

Доступность слабо размеченных данных

Этот случай изображен в центре на рис. 1.9. Некоторые датасеты содержат метки, которые не являются целевыми результатами модели, но в определенной мере имеют к ним отношение. Например, в случае с потоковым музыкальным проигрывателем история воспроизведенных и пропущенных песен может быть использована в качестве набора данных со слабыми метками, чтобы предсказывать предпочтения пользователя. Хотя пользователь не делает пометку о том, что ему не нравится определенная песня, пропуск песни может указывать на отказ от ее прослушивания. Слабые метки по определению обеспечивают меньший уровень точности, но зато такие датасеты обычно проще найти, чем наборы с точными метками.

Доступность неразмеченных данных

Этот случай показан справа на рис. 1.9. Иногда мы имеем набор входных данных без соответствующих меток, зато этот датасет содержит релевантные образцы. Например, в случае перевода текстов с одного языка на другой вы можете получить большие коллекции текста на обоих языках, между которыми не будет четко выраженного соответствия. Это означает, что нам потребуется либо добавить метки к набору данных, либо найти модель, способную обучаться на неразмеченных данных, либо даже объединить эти подходы.

Данные еще требуется собрать

Бывает так, что у нас нет даже неразмеченных данных и их еще только предстоит собрать. Хотя это нередко кажется непосильной задачей, к настоящему

тасет, который вы изучили и используете, дает ценную информацию для отбора следующей версии датасета и создания признаков для своих моделей.

Теперь давайте перейдем к нашему учебному примеру и применим полученные знания, чтобы выделить ряд возможных моделей и датасетов и выбрать наиболее подходящие.

Формулировка задачи по созданию редактора на основе МО

Давайте посмотрим, как можно продвигаться итерациями по юзкейсу продукта, чтобы правильно сформулировать задачу. Мы пошагово разберем процесс продвижения от цели продукта (помогать пользователям лучше формулировать вопросы) к парадигме МО.

Нам нужно создать редактор, который будет принимать любые вопросы, заданные пользователем, и улучшать их. Но что здесь следует понимать под улучшением? Для начала давайте более четко определим, в чем состоит цель продукта — помощника по написанию текстов.

Люди часто ищут ответы на свои вопросы на форумах, в социальных сетях и на сайтах вроде Stack Overflow¹. Однако в случае неудачной формулировки вопроса им не всегда удастся получить полезный ответ. При этом в проигрыше оказывается и тот пользователь, который непосредственно ищет ответ на вопрос, и те пользователи, которые могли бы воспользоваться готовым ответом, столкнувшись с аналогичной проблемой в будущем. Таким образом, наша цель будет состоять в *создании ассистента, помогающего пользователям лучше формулировать вопросы*.

Теперь, когда мы определились с целью продукта, необходимо решить, какой метод моделирования лучше использовать. Чтобы принять решение, мы должны пройти через уже упоминавшийся итерационный цикл выбора модели и валидации данных.

Делаем все с помощью МО: подход «от начала до конца»

Здесь словосочетание «от начала до конца» означает, что весь путь от входных данных до данных на выходе реализуется без каких-либо промежуточных этапов. В большинстве случаев цель продукта носит специфический характер, поэтому, чтобы реализовать юзкейс от начала до конца, зачастую требуется использовать индивидуально разработанную продвинутую МО-модель. Это решение подойдет

¹ <https://stackoverflow.com/>

для команд, у которых есть ресурсы для разработки и поддержки таких моделей, но чаще всего лучше начать с моделей попроще.

В нашем случае можно попытаться собрать датасет с плохо сформулированными вопросами и их профессионально отредактированными версиями. Это позволит нам получить новый текст напрямую из исходного, используя генеративную модель.

На рис. 1.10 показано, как это будет выглядеть на практике. Слева на этой простой схеме представлен пользовательский ввод, справа — необходимый результат, а модель располагается в середине.

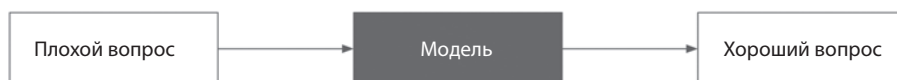


Рис. 1.10. Подход «от начала до конца»

Выбор этого подхода создает значительные проблемы:

Данные

Для получения необходимого датасета потребуется найти пары вопросов с одним и тем же смыслом, но с разным качеством формулировки. Мы вряд ли найдем этот датасет в готовом виде. Самостоятельное же создание такого датасета обойдется недешево, поскольку нам потребуется помощь профессиональных редакторов.

Модель

В последние годы наблюдается стремительный прогресс в разработке моделей, позволяющих преобразовать одну текстовую последовательность в другую, — генеративных моделей, о которых говорилось ранее. Sequence-to-sequence модели (из последовательности в последовательность), описанные в статье Суцкевера (Sutskever) и др. «Обучение sequence-to-sequence моделей с использованием нейронных сетей» («Sequence to Sequence Learning with Neural Networks»¹), были впервые предложены в 2014 году как решение для задач перевода с одного языка на другой, и теперь они заметно сокращают разрыв между машинным и человеческим переводом. Однако эти модели пока успешно справляются только с переводом отдельных предложений текста и почти никогда не используются для перевода более чем одного абзаца. Это объясняется тем, что они еще не научились хранить контекст нескольких абзацев. Кроме того, эти модели, как правило, имеют большое количество параметров, и потому их обучение происходит очень медленно. Это не про-

¹ <https://arxiv.org/abs/1409.3215>

блема, если обучение нужно провести лишь один раз, но если модель нужно переобучать буквально ежечасно или ежедневно, длительность обучения играет важную роль.

Время реакции

Модели «sequence-to-sequence» часто представляют собой *авторегрессионные модели*, в которых для перевода каждого последующего слова сначала требуется обработать предыдущее. Это позволяет им обрабатывать информацию, извлеченную из соседних слов, но замедляет процесс обучения и получения результата в отличие от более простых моделей, длительность задержки у которых не превышает долей секунды (вместо нескольких секунд). Вы можете оптимизировать такую модель, и она будет работать достаточно быстро, однако это потребует дополнительных инженерных усилий.

Простота реализации

Сложные модели состоят из множества «подвижных частей», поэтому их обучение — достаточно тонкий процесс, который подвержен ошибкам. Это означает, что мы должны учесть баланс между потенциальной производительностью такой модели и степенью сложности, которую она привносит в процесс разработки. Повышенная сложность не только замедлит процесс построения пайплайна, но и сделает более трудоемким техническое обслуживание модели. Если вы допускаете, что вашу модель потребуется вновь и вновь дорабатывать вашим товарищам по команде, возможно, будет лучше воспользоваться набором более простых и понятных моделей.

Несмотря на то что подход «от начала до конца» может быть эффективен, он не гарантирует успеха. К тому же этот подход требует значительной предварительной работы по сбору данных и инженерному проектированию, поэтому будет разумно изучить альтернативы, которые мы рассмотрим далее.

Простейший подход: «сам себе алгоритм»

Как вы увидите в интервью в конце этого раздела, специалисту по данным часто приходится самому *«стать алгоритмом»*, перед тем как приступить к его реализации. Иными словами, чтобы понять, какой способ автоматизации лучше использовать для вашей задачи, сначала попробуйте решить ее самостоятельно. Итак, если бы мы самостоятельно редактировали вопросы, чтобы улучшить их читаемость и увеличить шансы на получение ответа, то как бы мы делали это?

При таком подходе мы можем вообще не использовать данные, а вместо этого первым делом определить признаки хорошо написанного текста или вопроса. Можно составить общие рекомендации по написанию текстов, обратившись к профессиональному редактору или изучив руководства по газетным стилям.

Нам также потребуется лучше изучить отдельные образцы данных из набора, выделить имеющиеся тенденции и положить их в основу нашей стратегии моделирования. Мы не будем рассматривать это сейчас, поскольку эта тема подробно освещена в главе 4.

Для начала можно изучить результаты проведенных на сегодняшний день исследований¹ и выделить несколько признаков хорошего текста. К числу таких признаков относятся, например:

Простота текста

Начинающим писателям часто рекомендуют использовать более простую лексику и структуру предложения. Соответственно, мы могли бы определить ряд критериев в отношении надлежащей длины слов и предложений и рекомендовать внесение соответствующих изменений.

Тон текста

Мы можем оценивать частоту использования наречий, прилагательных в превосходной степени и знаков пунктуации для определения полярности текста. В зависимости от контекста, вопросы, заданные в категоричном тоне, могут получить меньше ответов.

Структурные признаки

Наконец, мы можем попытаться выделить такие важные структурные компоненты, как наличие приветствия или вопросительного знака.

После того как мы сгенерируем полезные признаки, можно будет реализовать простое решение для выдачи рекомендаций на их основе. Этот этап не подразумевает использования МО, но он играет критически важную роль: во-первых, на нем создают быстро реализуемое базовое решение (baseline), а во-вторых, он служит критерием для оценки моделей.

Затем, получив датасет с примерами «хорошего» и «плохого» текста, можно будет проверить наши интуитивные представления о хорошо написанном тексте и убедиться, что выбранные признаки позволяют отличать «хороший» текст от «плохого».

Золотая середина: обучение на полученном опыте

Теперь, когда у нас в распоряжении есть исходный набор признаков, мы можем попытаться использовать их для *обучения стиливой модели на массиве данных*. Это можно сделать, собрав датасет с признаками, о которых мы уже говорили, и обучив на нем классификатор различать хорошие и плохие образцы.

¹ <https://oreil.ly/jspYn>

Когда у нас есть модель, способная классифицировать письменный текст, мы можем провести ее анализ, чтобы определить, какие признаки позволяют наиболее уверенно предсказывать результат, и предлагать их в качестве рекомендаций. Как это можно сделать на практике, будет показано в главе 7.

На рис. 1.11 показан этот подход. Слева изображена модель, обученная классифицировать, хорошо или плохо поставлен вопрос. Справа показано, как обучаемая модель принимает вопрос и оценивает возможные варианты его перефразирования для получения более «хорошего» вопроса. Вариант, получивший наивысшую оценку, рекомендуется пользователю.

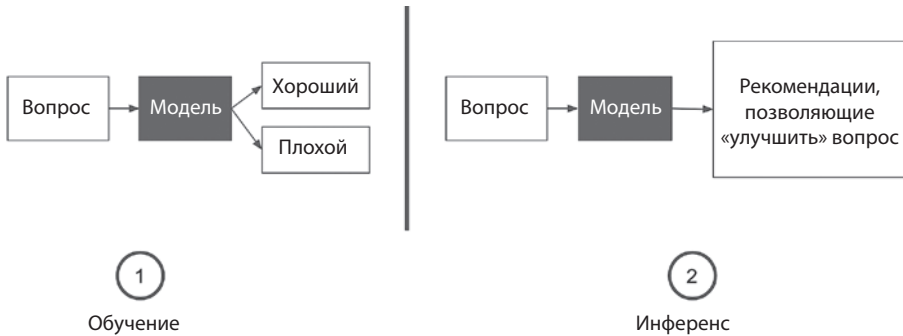


Рис. 1.11. Золотая середина между ручным подходом и подходом «от начала до конца»

Давайте еще раз рассмотрим трудности, перечисленные в разделе «Делаем все с помощью МО: подход “от начала до конца”» на с. 35, и посмотрим, облегчит ли работу классификатор:

Датасет

Мы можем получить датасет с хорошими и плохими образцами, собирая вопросы с интернет-форумов вместе с некоторыми показателями их качества, например количеством просмотров или оценок «нравится». В отличие от подхода «от начала до конца», здесь не нужно иметь «хорошие» и «плохие» версии одних и тех же вопросов. Вполне подойдет набор разных «хороших» и «плохих» вопросов, из которого можно выделить признаки. Такой датасет найти проще.

Модель

Здесь следует учитывать два момента: насколько высока точность предсказаний модели (насколько эффективно она различает хорошие и плохие тексты) и насколько легко из этих текстов извлечь признаки (можем ли мы понять, на основе каких атрибутов классифицируются образцы). Мы можем использовать разные модели и извлекать из текста разные наборы признаков, пока алгоритм не станет более объяснимым.

Время реакции

Большинство классификаторов текста работает достаточно быстро. Мы могли бы начать с простой модели, такой как «случайный лес» (random forest), способной возвращать результаты менее чем за десятую долю секунды на стандартном оборудовании, и при необходимости сменить архитектуру модели на более сложную.

Простота реализации

Поскольку классификация текста — более простой процесс по сравнению с генерированием текста, создать такую модель можно достаточно быстро. В интернете можно найти много примеров работоспособных пайплайнов классификации текста, и множество таких моделей уже используются в реальных проектах.

Начав с эвристической оценки и создав простую модель, мы сможем быстро получить базовое решение (baseline) и таким образом сделать первый шаг к решению задачи. Более того, эта исходная модель позволит нам понять, что следует делать дальше (подробнее об этом будет рассказано в части III).

О том, почему важно начинать с baseline, я поговорил с Моникой Рогати (Monica Rogati). В интервью она поделилась опытом, полученным в то время, когда она помогала создавать продукты командам по обработке данных.

Моника Рогати: как выбирать и приоритизировать МО-проекты

После окончания аспирантуры в области компьютерных наук Моника Рогати начала карьеру в компании LinkedIn, где приняла участие в разработке основных продуктов. В частности, она интегрировала МО в алгоритм выдачи рекомендаций «Люди, которых вы можете знать» и работала над созданием первой версии системы сопоставления вакансий и соискателей работы. Затем она заняла пост вице-президента по обработке данных в компании Jawbone, где ей довелось создать и возглавить целое подразделение по обработке данных. В настоящее время Моника консультирует десятки компаний с численностью персонала от 5 до 8 тысяч. Она была рада поделиться со мной некоторыми советами по проектированию и реализации МО-продуктов.

Вопрос. Как вы оцениваете масштаб работы при создании продукта на базе МО?

Ответ. Мы всегда должны стремиться использовать для решения задачи наиболее подходящие инструменты и применять МО только тогда, когда это действительно нужно.

Допустим, вы хотите предсказывать последующие действия пользователя приложения и отображать для них подсказки. В таком случае следует начать с обсуждения модели и продукта. Помимо прочего, при проектировании продукта следует подумать о том, как можно обеспечить бесперебойность работы в случае отказа МО.

Для начала можно принять в расчет то, насколько вы уверены в выдаваемых моделью предсказаниях. Это позволит по-разному отображать подсказки в зависимости от степени уверенности. Когда степень уверенности больше 90%, тут же на видном месте появится подсказка. Если степень уверенности больше 50%, то подсказка будет не столь заметной. А когда степень уверенности меньше 50%, то никакой подсказки не будет вовсе.

Вопрос. Как вы принимаете решение о том, на чем сосредоточиться при работе над проектом?

Ответ. Вы должны найти *узкое место*, то есть тот элемент пайплайна, оптимизация которого может дать наибольший эффект. В ходе своей работы с компаниями я часто сталкиваюсь с ситуацией, когда они направляют усилия не на ту проблему или работают над проблемой, для решения которой еще не достигнута подходящая стадия роста.

Часто проблемы связаны с используемой моделью. Лучший способ выявления таких проблем состоит в замене модели чем-то попроще и отладке всего пайплайна. Часто проблемы не связаны с точностью модели: продукт терпит крах, даже несмотря на успешную модель.

Вопрос. Почему вы обычно рекомендуете начинать с простой модели?

Ответ. При составлении плана наша цель — исключить риски, связанные с моделью. Лучший способ это сделать состоит в том, чтобы сначала создать baseline для оценки худшего сценария производительности. В случае с нашим примером мы можем просто давать пользователю подсказку вне зависимости от его предыдущих действий.

Как часто предсказание модели будет точным, если выбрать такое решение? Насколько сильно будет раздражать пользователя наша модель в случае неверного предсказания? Будет ли по-прежнему полезен наш продукт, если модель будет работать не намного лучше, чем это базовое решение?

Эти вопросы вполне уместны при обработке и генерировании текста на естественном языке в таких системах, как чат-боты, системы языкового перевода, системы вопросов и ответов и системы автоматического реферирования. Так, в случае систем автоматического реферирования простое извлечение основной части, используемых в статье ключевых слов и категорий закрывает потребности большинства пользователей.

Вопрос. Как найти узкое место, когда пайплайн уже реализован?

Ответ. Сначала нужно представить, будто узкое место уже улучшено, и спросить себя, стоило ли это затраченных усилий. Обычно еще до начала работы над проектом я предлагаю специалистам по обработке данных написать твит, а руководству компании — пресс-релиз. Это позволяет исключить вероятность выполнения какой-либо работы просто потому, что это было бы круто, а также оценить, стоит ли ожидаемый эффект затрачиваемых усилий.

В идеале можно оценить результаты независимо от исхода. Произведет ли работа какой-либо эффект даже в том случае, если ее результаты будут не лучшими? Научила ли она вас чему-то или подтвердила ли какие-то предположения?

На сайте LinkedIn у нас был очень полезный элемент дизайна — небольшое окно с несколькими строками текста и гиперссылками, которое можно было настраивать с помощью наших данных. Это упрощало проведение экспериментов в таких проектах, как выдача рекомендаций по вакансиям, потому что дизайн уже был одобрен. Поскольку мы почти не затрачивали ресурсы, эффект внесенных изменений был невелик, что позволяло сократить итерационный цикл. В результате основным препятствием становились уже не технические соображения: вопросы этики, равномерное распределение качества обслуживания и брендинг.

Вопрос. Как вы принимаете решение о том, какие методы моделирования использовать?

Ответ. Первая линия обороны здесь — внимательное ознакомление с данными. Допустим, нам нужно создать модель, рекомендующую пользователям группы в социальной сети LinkedIn. Простейший подход состоит в выдаче рекомендации наиболее популярной группы, в названии которой присутствует название компании, где работает пользователь. Однако когда мы ознакомились с рядом примеров, выяснилось, что в случае компании Oracle одна из популярных групп называлась «Oracle sucks!» («Oracle — отстой!»), а такую группу не очень этично рекомендовать сотрудникам компании, в которой они работают.

Всегда полезно потратить усилия на то, чтобы своими глазами рассмотреть входные и выходные данные вашей модели. Проведите разбор нескольких примеров и убедитесь, что в них нет ничего странного. Руководитель моего отдела в компании IBM всегда настаивал, чтобы перед началом реализации определенной функции мы в течение часа попробовали сделать то же самое вручную.

Внимательное ознакомление с данными поможет вам продумать эвристические алгоритмы, модели и способы формулировки цели продукта. Если вы распределите образцы из вашего датасета по степени распространенности, возможно, вы сможете сразу определить юзкейс и даже разметить 80% данных.

Например, в компании Jawbone у сотрудников было принято вводить «фразы», когда они вели журнал своего рациона питания. К тому моменту, как мы вручную пометили 100 самых распространенных образцов, что составляло 80% фраз,

мы уже четко понимали, что представляют собой основные проблемы. Одна из них — разнообразие используемых кодировок текста и языков.

Последняя линия обороны — это уже изучение результатов сотрудниками разных подразделений. Это позволит вам выявить случаи, в которых, например, модель проявляет дискриминационное поведение: скажем, называет друзей пользователя «обезьянами» или бестактно напоминает ему о каких-то неприятных событиях из прошлого при работе интеллектуальной функции ретроспективного взгляда по типу «В это же время в прошлом году».

Закключение

Как мы выяснили, создание приложения на базе МО начинается с оценки обоснованности применения МО и выбора подхода. В большинстве случаев проще всего начать с попытки применить метод обучения с учителем. Среди прочих алгоритмов на практике также часто встречаются модели классификации, извлечения знаний, каталогизации и генеративные модели.

При выборе подхода следует оценить, насколько трудно будет собрать размеченные, слабо размеченные или какие-либо данные вообще. Затем, после определения цели продукта, необходимо сравнить потенциальные модели и датасеты и выбрать тот подход к моделированию, который наилучшим образом позволяет реализовать эту цель.

Мы посмотрели, как можно выполнить эти шаги при разработке МО-редактора, выбрав для него в качестве отправной точки простой эвристический алгоритм и классификационный подход к обучению. И наконец, из интервью с Моникой Рогати мы узнали, каким образом ведущие специалисты применяют эту практику для успешной поставки МО-моделей пользователям.

Теперь, когда мы уже выбрали базовый подход, пришло время определить метрики успешности и составить план действий, способный обеспечить нам стабильное продвижение вперед. Этот план будет включать в себя определение минимальных требований к производительности, изучение доступных ресурсов моделирования и источников данных, а также создание простого прототипа.

Все вышеперечисленное мы рассмотрим в главе 2.

ГЛАВА 2

Составление плана

В предыдущей главе мы узнали, как оценить, нужно ли использовать МО и в каких случаях, а также как из цели продукта сформировать адекватное задание для МО. В этой главе мы поговорим об использовании метрик для отслеживания прогресса разработки и машинного обучения, а также сравним различные реализации МО. После этого мы определим методы создания baseline (базового решения) и составим план итеративной доработки модели.

Мне неоднократно приходилось видеть, как проекты МО с самого начала были обречены на неудачу из-за несоответствия между метриками продукта и модели. Проекты часто терпят фиаско не из-за низкой производительности модели, а из-за того, что вполне эффективная модель не несет никакой пользы продукту. Именно поэтому я решил посвятить целую главу метрикам и планированию.

Мы дадим рекомендации, как с учетом имеющихся ресурсов и ограничений составить план действий, который существенно упростит работу над МО-проектом.

Для начала давайте подробно разберемся в том, что собой представляют метрики производительности.

Оценка успешности

При применении МО начинать нужно с самой простой модели, отвечающей требованиям продукта, поскольку наиболее быстрый способ продвижения вперед в процессе МО — получать и анализировать результаты. В предыдущей главе мы рассмотрели три способа реализации МО-редактора, различающиеся по сложности. Давайте еще раз вспомним, что они собой представляют.

Baseline: проектирование эвристического алгоритма на основе знаний о предметной области

Для начала, имея представление о хорошо написанном тексте, мы просто самостоятельно определяем правила. Проверить эти правила можно, посмотрев, позволяют ли они различать хорошо и плохо написанный текст.

Простая модель: классификация текста на «хороший» или «плохой» и генерация рекомендаций с помощью классификатора

Затем мы можем научить простую модель различать хорошо и плохо сформулированные вопросы. Добившись высокой производительности этой модели, мы можем посмотреть, какие признаки позволяют с высокой уверенностью предсказывать хорошо написанный вопрос, и использовать их в качестве рекомендаций.

Сложная модель: обучение модели «от начала до конца»; модель преобразует плохо написанный текст в хороший

Этот подход наиболее сложен в плане как модели, так и данных. Однако если есть ресурсы, чтобы собрать обучающие данные, а затем создать и поддерживать сложную модель, мы сможем полностью удовлетворять требования к продукту.

Каждый из этих подходов имеет свои особенности и по мере накопления опыта может быть усовершенствован. Однако при работе над МО всегда следует определять общий набор метрик для сравнения успешности пайплайнов моделирования.



Не всегда в МО есть необходимость

Вероятно, вы заметили, что при создании baseline мы совсем не полагаемся на МО. Как уже говорилось в главе 1, некоторые функции можно реализовать и без него. Также важно понимать, что даже если реализуемая функция может выиграть от применения МО, в качестве первой версии часто можно использовать простой эвристический алгоритм. Опробовав эвристический алгоритм на практике, вы даже можете решить, что вам в принципе не требуется МО.

Написание эвристического алгоритма часто является самой быстрой реализацией функциональности. После того как вы реализуете и попытаетесь применить новую функцию, у вас сложится более четкое понимание потребностей пользователя. Это понимание позволит вам определить, нужно ли МО, и выбрать подходящий метод моделирования.

В большинстве случаев создание исходной версии без МО — самый быстрый способ разработки МО-продукта.

Итак, рассмотрим четыре категории метрик производительности, от которых в значительной мере зависит польза МО-продукта: бизнес-метрики, метрики модели, метрики актуальности данных и метрики скорости. Хорошо понимая эти метрики, мы сможем точно оценивать производительность каждой итерации.

Производительность с точки зрения бизнеса

Ранее мы говорили, что в первую очередь важно поставить четкую цель продукта или функции. Сделав это, необходимо определить показатель успешности результата. Эта метрика должна определяться независимо от

прочих метрик модели и отражать только степень успешности продукта. Продуктовые метрики могут представлять собой простой показатель, например количество пользователей, которым нравится новая функция, или более тонкий показатель, такой как коэффициент кликабельности (CTR) наших рекомендаций.

В конечном итоге для нас важны только продуктовые метрики, поскольку они отражают цели нашего продукта или новой функции. Все остальные метрики должны использоваться лишь как инструмент для улучшения продуктовых метрик, однако и ими не стоит ограничиваться. Хотя цель большинства проектов — улучшение какой-то одной продуктовой метрики, результат оценивается все же по нескольким метрикам, в том числе по *метрикам ограждения (guardrail metrics)*, которые не должны падать ниже определенного уровня. Так, цель МО-проекта может состоять в улучшении какой-то одной метрики при сохранении других, например повышении CTR при сохранении средней длительности пользовательского сеанса.

Для МО-редактора мы можем подобрать метрику, отражающую степень полезности рекомендаций, например, это может быть доля случаев, когда пользователь следует рекомендации. Чтобы вычислить такую метрику, можно фиксировать, понравился ли пользователю предложенный вариант, отображая его поверх поля ввода и делая кликабельным.

Мы знаем, что любой продукт может быть разработан с помощью разных подходов к МО. Для оценки эффективности каждого из подходов нужно оценивать производительность модели.

Производительность модели

Для большинства онлайн-продуктов основной метрикой успешности модели является доля посетителей, использующих ее результаты, в отношении к общему числу посетителей. Так, например, производительность системы рекомендаций часто оценивается по количеству пользователей, перешедших по ссылке на рекомендуемые продукты (возможные проблемы этого подхода будут рассмотрены в главе 8).

Однако вы не сможете получить данные по этим метрикам до релиза продукта. Чтобы все-таки оценивать прогресс, необходимо определить отдельную метрику, которую называют *офлайн-метрикой*, или *метрикой модели*. Хорошая офлайн-метрика должна измеряться без взаимодействия с пользователями и как можно сильнее коррелировать с метриками и целями продукта.

При разных подходах к моделированию используются разные метрики модели, и смена подхода может существенно упростить достижение необходимой производительности.

Допустим, вы хотите, чтобы при вводе поискового запроса в онлайн-магазине отображались полезные подсказки. Оценить успешность этой функции можно по количеству переходов по ссылкам, то есть по тому, насколько часто пользователи нажимают на отображаемую подсказку.

Для генерирования таких подсказок можно создать модель, которая по мере ввода текста будет давать подсказки, предвосхищая запрос пользователя. Оценить производительность этой модели можно, измерив ее точность на уровне слов, то есть посчитав, насколько часто и правильно она предсказывает последующие слова. Чтобы обеспечить высокий CTR, такая модель должна работать чрезвычайно точно, ведь подсказку могут посчитать бесполезной при ошибке хотя бы в одном слове. Слева на рис. 2.1 изображена схема этого подхода.

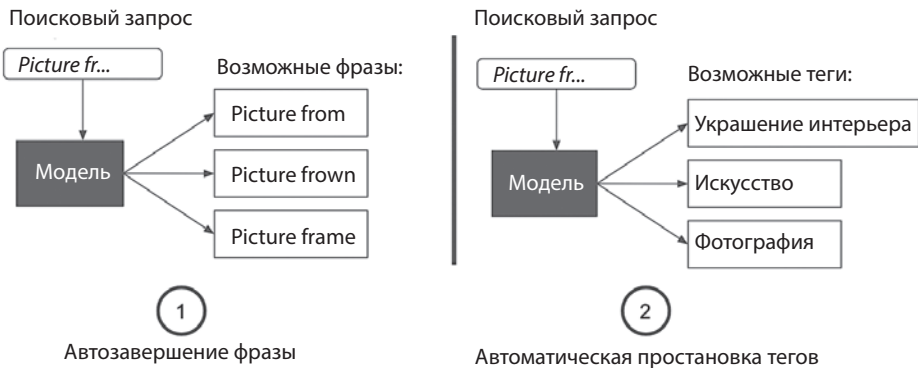


Рис. 2.1. Немного изменив продукт, можно существенно упростить моделирование (Picture from — Картина из...; Picture frown — Картина хмурого...; Picture frame — Рамка для фото)

Альтернативный подход может состоять в том, чтобы обучить модель классифицировать пользовательский ввод согласно категориям вашего каталога и предлагать три категории, которые с наибольшей вероятностью подходят пользователю. Производительность модели будет оцениваться по точности предсказания всех категорий, а не каждого слова. Поскольку количество категорий в каталоге гораздо меньше количества существующих в языке слов, эта метрика модели будет гораздо легче поддаваться оптимизации. Кроме того, чтобы сгенерировать ссылку, этой модели достаточно точно предсказать хотя бы одну категорию. С такой моделью гораздо проще повысить CTR продукта. Схема этого подхода представлена справа на рис. 2.1.

Как видите, незначительная модификация взаимодействия между моделью и продуктом часто позволяет применять более прямолинейный подход к моделированию и получать более надежные результаты. Вот еще несколько примеров модификации приложения для упрощения моделирования.

- *Изменить интерфейс так, чтобы не учитывались предсказания модели, степень уверенности в которых будет ниже определенного порогового значения.* Например, во время разработки модель для автозавершения вводимой пользователем фразы может эффективно работать только на ограниченном подмножестве фраз. Учитывая это, можно реализовать логику так, чтобы подсказки отображались пользователю только тогда, когда уверенность модели в предсказании будет превышать 90%.
- *Предоставлять несколько дополнительных предсказаний или эвристик.* Например, на большинстве веб-сайтов отображается сразу несколько рекомендаций, предлагаемых моделью. Отображение пяти возможных вариантов вместо одного повысит вероятность того, что посетители воспользуются рекомендациями, даже если модель та же самая.
- *Уведомлять пользователей о том, что модель еще находится на этапе тестирования, и давать им возможность обратной связи.* Автоматически определяя, что язык для пользователя не родной, и делая перевод, веб-сайты часто отображают кнопку, с помощью которой пользователи могут сообщить, был ли перевод точным и полезным.

Даже если используемый подход к моделированию хорошо соответствует решаемой задаче, иногда стоит сгенерировать дополнительные метрики модели, которые будут сильнее коррелировать со степенью эффективности продукта.

Мне как-то довелось работать со специалистом по обработке данных, который создал модель для генерирования HTML-кода на основе нарисованных от руки эскизов простых веб-сайтов (см. его статью в блоге «Автоматизированная разработка фронтенда с помощью глубокого обучения» («Automated Front-End Development Using Deep Learning»¹)). Метрика оптимизации этой модели каждый раз сравнивает предсказание HTML-токена² с правильным, основываясь на функции потери кросс-энтропии³. Однако цель продукта здесь состоит в том, чтобы сгенерированный HTML-код выглядел в итоге как веб-сайт с исходного эскиза, вне зависимости от порядка токенов.

Кросс-энтропия не учитывает выравнивания: если модель сгенерирует правильный набор HTML-токенов, за исключением одного лишнего токена в начале, то все токены будут сдвинуты на одну позицию по сравнению с целевым результатом. Несмотря на то что это почти идеальный результат, значение энтропии будет очень высоким. Это означает, что при оценке полезности модели мы не должны ограничиваться метриками оптимизации. В данном случае лучшей мерой сте-

¹ <https://oreil.ly/SdYQj>

² Контейнер, состоящий из пары дескрипторов: открывающего и закрывающего тега. — *Примеч. ред.*

³ Логарифмическая функция потерь; функция потерь перекрестной энтропии. — *Примеч. ред.*

пени сходства между сгенерированным и необходимым HTML-кодом будет метрика двуязычной оценки дублера BLEU (Bilingual Evaluation Understudy)¹.

Наконец, при проектировании продукта следует исходить из разумных предположений о степени эффективности модели. Когда продукт полагается на то, что модель будет идеальной, это часто ведет к получению неточных или даже опасных результатов.

Допустим, что вам нужно создать модель, способную по фотографии таблеток сообщить пользователю название и дозировку этого препарата. При какой минимальной точности такая модель еще будет оставаться полезной? Если такое требование к точности трудно удовлетворить с помощью существующих методов, можете ли вы перепроектировать продукт так, чтобы он был полезен пользователям и не подвергал их здоровью риску в случае ошибочных предсказаний?

Наш учебный пример должен выдавать рекомендации по написанию текста. Для большинства МО-моделей есть входные данные, с которыми они справятся хорошо, и данные, с которыми возникнут трудности. На уровне продукта мы должны позаботиться о том, чтобы в тех случаях, когда мы не можем помочь, мы хотя бы не делали хуже, то есть чтобы результат вообще не выдавался в тех случаях, когда он хуже входных данных (текста пользователя). Как это можно выразить в метриках модели?

Допустим, что нам нужно создать классификационную модель, предсказывающую качество вопроса по количеству отметок «нравится». Метрика *precision* (точность) этого классификатора может быть выражена как доля верно определенных хороших вопросов среди всех вопросов, отнесенных к категории хороших, а метрика *recall* (полнота) — как доля вопросов, относимых к категории хороших, среди всех хороших вопросов датасета.

Чтобы всегда получать релевантные рекомендации, нам потребуется отдать приоритет метрике *precision*, поскольку когда высокоточная модель классифицирует вопрос как хороший (с выдачей соответствующей рекомендации), он с высокой степенью вероятности действительно является хорошим, то есть при высокой точности выдаваемая нами рекомендация, как правило, является правильной. Подробнее о том, почему для выдачи рекомендаций лучше всего подходят высокоточные модели, можно прочитать в интервью с Крисом Харландом (Chris Harland) на с. 214.

Измеряются такие метрики на результатах, выдаваемых моделью на репрезентативном валидационном наборе. О том, что под этим понимается, подробно рассказано в разделе «Оценка модели: не ограничивайтесь точностью» на с. 140. На данном этапе будем считать, что это часть датасета, которая не используется

¹ <https://oreil.ly/8s9JE>

при обучении. С ее помощью оценивается производительность модели на незнакомых данных.

Наряду с начальной производительностью модели большую роль играет и ее способность оставаться полезной при изменении поведения пользователей. Модель, обученная на определенном датасете, будет хорошо справляться с обработкой аналогичных данных, однако как мы узнаем о том, что пришла пора обновить датасет?

Актуальность данных и сдвиг распределения

Модели машинного обучения с учителем получают прогностическую силу за счет изучения корреляций между признаками входных данных и предсказываемыми целевыми показателями. Это означает, что для того, чтобы модель хорошо работала на определенных входных данных, она должна быть обучена на аналогичных обучающих данных. Модель, обученная предсказывать возраст пользователя исключительно на фотографиях мужчин, будет плохо работать на фотографиях женщин. Однако даже когда обучение модели производится на адекватном датасете, распределение обрабатываемых данных часто изменяется с течением времени. В случае такого *сдвига* распределения данных часто требуется соответственно изменить и модель, чтобы сохранить прежний уровень производительности.

Допустим, что, обратив внимание на то, как дождь влияет на дорожное движение в Сан-Франциско, вы решили создать модель для предсказания дорожных условий на основе количества осадков, выпавших за прошедшую неделю. Если вы создадите свою модель в октябре, используя данные за прошедшие три месяца, то модель будет обучена на данных с ежедневным уровнем осадков менее 1 дюйма. Как может выглядеть такое распределение, показано на рис. 2.2. Однако по мере приближения зимы средний уровень осадков возрастет до 3 дюймов, что превысит показатели всех примеров обучающего датасета (см. рис. 2.2.) Если при этом вы не дообучите модель на более свежих данных, то она вряд ли будет работать хорошо.

Обычно модель может хорошо работать на незнакомых ей данных лишь в том случае, если они достаточно похожи на данные, на которых она обучалась.

Разные задачи могут иметь разные требования к актуальности данных. Если в случае сервиса перевода с древних языков можно рассчитывать на то, что данные практически не будут изменяться, то в случае поисковой системы следует предполагать, что данные будут изменяться по мере изменения поисковых привычек пользователей.

Оцените, каких усилий потребует поддержание вашей модели *в актуальном состоянии*. Насколько часто потребуется дообучать модель и каких затрат потребует каждая такая процедура?

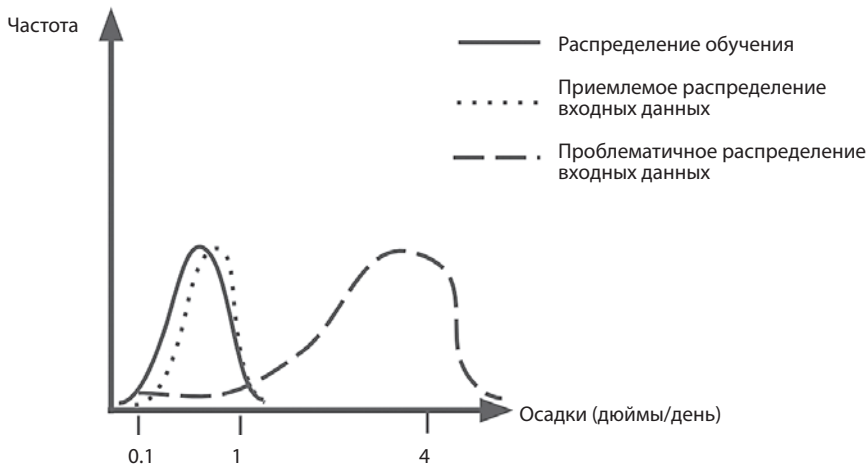


Рис. 2.2. Изменение распределений

Что касается МО-редактора, представления о хорошо написанном тексте изменяются сравнительно медленно, может, раз в год. В то же время требования к актуальности данных будут сильно варьироваться, если мы сосредоточимся на отдельных предметных областях. Так, например, представления о правильной формулировке вопросов в области математики будут меняться гораздо медленнее, чем представления о хорошо составленном вопросе о музыкальных трендах. Таким образом, по нашей оценке, потребуется дообучать модель с периодичностью раз в год и, соответственно, собирать актуальные данные для проведения этого дообучения.

При использовании базового решения или простой модели не требуются парные данные, что сильно упрощает процесс сбора данных (нам нужно будет просто собирать появившиеся за год новые вопросы). Однако при использовании сложной модели требуются парные данные, а значит, нам нужно будет ежегодно собирать примеры «хорошей» и «плохой» формулировки одних и тех же фраз. Таким образом, соблюдать установленные нами требования к актуальности данных гораздо труднее в случае модели, требующей использования парных данных, поскольку обновление такого датасета — более трудоемкий процесс.

Задача сбора данных часто упрощается, когда приложение становится популярным. Если бы наш сервис формулировки вопросов набрал большую популярность, мы могли бы добавить кнопку, позволяющую пользователям оценивать качество результатов. После этого мы могли бы собрать обработанные за год входные данные вместе с соответствующими предсказаниями модели и оценками пользователей и использовать эти данные в качестве обучающего датасета.

Однако чтобы стать популярным, приложение должно быть полезным. В большинстве случаев это подразумевает своевременное реагирование на запросы пользователей. Соответственно, помимо прочего, следует принять во внимание и такой фактор, как скорость выдачи предсказаний моделью.

Скорость

В идеальном случае модель должна выдавать предсказание быстро. Это позволило бы одновременно обслуживать большое количество пользователей и упростило их взаимодействие с моделью. Так насколько быстро должна работать модель? В некоторых случаях пользователи будут рассчитывать на немедленное получение ответа, например при переводе небольших фраз, а в других — готовы подождать даже целые сутки для получения точного результата, например медицинского диагноза.

В нашем случае можно рассмотреть два способа выдачи рекомендаций. Первый — текстовое поле, где пользователь вводит текст и получает результат, щелкнув по кнопке подтверждения. Второй способ — выполнять динамическое обновление после ввода пользователем каждой новой буквы. Хотя второй вариант выглядит предпочтительнее, поскольку обеспечивает большую степень интерактивности, он более требователен к быстродействию модели.

При использовании кнопки подтверждения вполне допустима задержка в одну-две секунды. Однако для модели, выполняющей действия в процессе ввода текста, время реакции должно быть гораздо меньше секунды. Поскольку сложным моделям требуется больше времени на обработку данных, мы будем учитывать это требование в ходе итеративной доработки моделей. При использовании любой модели время прохождения единицы данных через весь пайплайн не должно превышать двух секунд.

Время исполнения возрастает по мере усложнения модели. Эта разница может быть ощутимой, даже если размер отдельных элементов данных сравнительно небольшой, как в задачах по обработке естественного языка (в отличие, например, от задач по обработке потокового видео). Так, для нашего учебного примера LSTM-модель¹ работает примерно в 3 раза медленнее, чем модель «случайный лес» (время выполнения соответственно составляет 22 и 7 мс). Эта разница может быть сравнительно небольшой при обработке отдельных элементов данных, но быстро возрастает, когда требуется выполнять инференс сразу для десятков тысяч образцов.

В некоторых сложных приложениях с инференсом связано множество сетевых вызовов и обращений к базе данных. При этом время реакции модели является сравнительно небольшим по сравнению с временем выполнения остальной логики приложения и уже не играет большой роли.

¹ LSTM — long short-term memory, дословно — «долгая краткосрочная память». — *Примеч. ред.*

В зависимости от специфики решаемой задачи, часто на первый план выходят некоторые другие соображения, например ограничения оборудования, длительность разработки или простота поддержки. Поэтому важно составить четкое представление о своих потребностях до выбора модели, чтобы этот выбор делался вполне осознанно.

Теперь, когда мы уже определили требования и соответствующие метрики, пришло время приступить к составлению плана. Для этого нам потребуется оценить сложность стоящих перед нами задач. Далее я покажу, как использовать результаты проведенной ранее работы и как, изучив датасет, принять решение о дальнейших шагах.

Оценка масштаба и возможных проблем

Как мы выяснили, эффективность МО часто оценивается по метрикам модели. Хотя эти метрики могут быть полезны сами по себе, их следует использовать для улучшения отдельных продуктовых метрик. По мере итеративной доработки пайплайна всегда нужно помнить о продуктовых метриках и стремиться их оптимизировать.

Мы уже узнали, как можно оценить осуществимость проекта и отслеживать прогресс. Логично, что следующим шагом будет составление плана реализации проекта, чтобы оценить его масштаб и срок выполнения, а также возможные препятствия в реализации.

Для успеха в МО вы должны понимать контекст задачи, иметь хороший датасет и подходящую модель.

В следующем разделе мы подробно рассмотрим каждую из этих составляющих.

Накапливайте знания в предметной области

Для простейшей исходной модели можно взять эвристический алгоритм — надежное эмпирическое правило, выработанное на основе имеющихся знаний о решаемой задаче и используемых данных. Разработку такого эвристического алгоритма лучше всего начать с изучения методов решения, используемых специалистами в предметной области. Большинство практически ориентированных приложений на самом деле не делают чего-то совершенно нового. Поэтому узнайте, как вашу задачу обычно решают специалисты.

Еще один хороший подход состоит в том, чтобы внимательно изучить используемые данные. Учитывая особенности вашего датасета, как бы вы решили задачу, если бы делали это вручную?

Таким образом, для выработки хорошего эвристического алгоритма я рекомендую либо изучить опыт специалистов в предметной области, либо внимательно рассмотреть используемые данные. Давайте поговорим об этих подходах чуть подробнее.

Изучение опыта специалистов

При внедрении автоматизации изучение опыта специалистов в соответствующей предметной области часто позволяет сэкономить десятки часов. Так, если бы мы решили создать систему для профилактического технического обслуживания (ТО) оборудования завода, мы могли бы для начала поговорить с руководителем завода и выяснить, из каких разумных допущений можно исходить в данном случае. Мы могли бы выяснить, с какой периодичностью на заводе проводят ТО, какие признаки обычно указывают на то, что станок скоро потребует ТО, какие нормативные требования предъявляются к ТО, и т. д.

Конечно, иногда найти эксперта достаточно сложно, как, например, в случае проприетарных данных для нового юзкейса для предсказания востребованности уникального элемента веб-сайта. Однако в таких случаях можно найти профессионалов, способных поделиться опытом решения сходных задач.

Изучение опыта специалистов поможет нам узнать, какие полезные признаки мы могли бы использовать и каких подводных камней нам следует избегать, и, самое главное, позволит не изобретать велосипед, чем часто грешат специалисты по обработке данных.

Изучение данных

И Моника Рогати, и Роберт Манро (Robert Munro) в своих интервью упоминают о том, что перед тем, как приступить к моделированию, важно внимательно изучить данные (см. раздел «Моника Рогати: как выбирать и приоритизировать МО-проекты» на с. 40 и раздел «Роберт Манро: как находить, размечать и использовать данные» на с. 118).

При создании любого информационного продукта важную роль играет исследовательский, или разведочный, анализ данных (Exploratory Data Analysis, EDA) — процесс визуализации и изучения датасета, часто с целью получить более четкое представление о решаемой бизнес-задаче. В дополнение к EDA также важно попробовать вручную разметить отдельные образцы так, как это должна делать модель. Это позволит проверить сделанные допущения и убедиться в том, что выбранная вами модель способна надлежащим образом использовать ваш датасет.

Проведя EDA, вы получите четкое понимание имеющихся в данных трендов, а разметив их, сможете быстрее создать эвристические алгоритмы для решения вашей задачи. Выполнив оба этих шага, вы будете лучше понимать, какие разно-

видности моделей хорошо подходят для вашего случая и какие дополнительные стратегии сбора данных и разметки могут потребоваться.

Следующим логичным шагом будет изучение опыта предшественников в решении сходных задач моделирования.

Используйте опыт предшественников

Решал ли уже кто-то до вас сходные задачи? Если это так, то лучший способ начать работу — изучить и воспроизвести полученные до вас результаты. Попробуйте найти общедоступные реализации, в которых используются сходные модели или сходные датасеты или и то и другое.

Идеальный вариант — найти открытый исходный код и свободно доступный датасет, хотя такое случается достаточно редко, особенно в случае очень специфических продуктов. Однако в любом случае самый быстрый способ начать работу над проектом МО — воспроизвести достигнутые ранее результаты и как-либо их доработать.

В сфере МО, имеющей так много «подвижных элементов», крайне важно опираться на опыт предшественников.



Если вы планируете использовать в своей работе открытый исходный код или открытые датасеты, убедитесь, что вам разрешено это делать. Большинство репозиторий и датасетов обладает лицензией, определяющей правила допустимого использования. Также не забудьте отдать должное авторам используемых вами ресурсов, желательно в виде ссылки на оригинальный источник.

Перед тем как вкладывать значительные ресурсы в реализацию проекта, полезно построить убедительное доказательство концепции. Так, например, перед тем как тратить время и деньги на разметку данных, необходимо убедиться в том, что вы сможете создать модель, способную обучаться на таких данных.

Так как же найти эффективный способ начать работу? И как обычно, ответ на этот вопрос имеет две составляющие — данные и код.

Открытые данные

Хотя вам не всегда удастся найти необходимый датасет, при отсутствии такового часто можно найти похожий датасет, который тоже будет полезен. Однако что в данном контексте следует понимать под «похожим»? Здесь уместно вспомнить, что МО-модели сопоставляют входные и выходные данные. Исходя из этого, можно сделать вывод, что похожий датасет — это датасет с похожими входными и выходными данными (но необязательно из той же предметной области).

Модели с похожими входными и выходными данными часто применяются в совершенно разных контекстах. Слева на рис. 2.3 показаны две модели, каждая из которых предсказывает текстовую последовательность на основе изображения. При этом одна модель генерирует описания фотографии, а вторая — HTML-код веб-страницы на основе соответствующего скриншота. Точно так же справа на рис. 2.3 показаны две модели, одна из которых предсказывает тип заведения общественного питания на основе текстового описания на естественном языке, а вторая — музыкальный жанр на основе партитуры музыкального произведения.

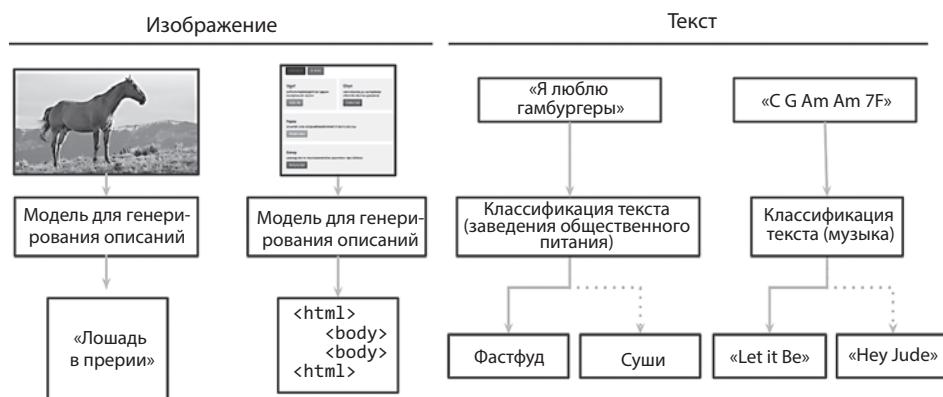


Рис. 2.3. Примеры моделей с аналогичными входными и выходными данными

В качестве примера давайте предположим, что нам нужно создать модель для предсказания размера аудитории новостных статей, но мы не можем найти датасет, содержащий новостные статьи вместе с соответствующими данными о количестве читателей. В таком случае можно для начала обучить модель на общедоступном датасете со статистикой посещения страниц Википедии¹. Если нам удастся обеспечить достаточную производительность, можно будет предположить, что наша модель сможет хорошо работать после того, как мы получим датасет с цифрами по количеству просмотров новостных статей. Таким образом, использование сходного датасета часто позволяет проверить действенность выбранного подхода и тем самым сделать более оправданными затраты ресурсов на сбор данных.

Этот метод будет уместен и в случае работы над проприетарными данными. Зачастую непросто найти нужный датасет для реализации задачи предсказания. Иногда нужные вам данные вообще не собираются. В таких случаях создание модели, способной хорошо работать на схожем датасете, является лучшим спо-

¹ <https://oreil.ly/PdwnG>

собом убедить стейкхолдеров, что необходим новый пайплайн сбора данных или упрощение доступа к существующему датасету.

Что касается общедоступных данных, в настоящее время регулярно появляются новые источники данных и коллекции. В частности, я могу порекомендовать:

- «Архив интернета» (Internet archive)¹ содержит множество датасетов, включая копии веб-сайтов, видеозаписи и книги.
- Раздел `r/datasets`² социальной сети Reddit посвящен обмену ссылками на датасеты.
- Большую коллекцию данных, относящихся к разнообразным предметным областям, можно найти в разделе `Datasets`³ социальной сети Kaggle.
- Огромным источником датасетов для МО является репозиторий UCI Machine Learning Repository⁴.
- Множество доступных датасетов можно найти с помощью приложения поиска датасетов⁵ от Google.
- Организация Common Crawl⁶ собирает данные из интернета в виде общедоступных архивов.
- Большую и постоянно пополняемую коллекцию датасетов для исследований в области МО также предлагает Википедия⁷.

В большинстве случаев хотя бы в одном из этих источников вы найдете датасет, схожий с нужным вам набором данных.

Обучив модель на этом *касательном* (*tangential*) *датасете*, вы сможете быстро создать прототип и проверить получаемые результаты. В некоторых случаях после обучения модели на касательном датасете можно даже частично перенести его эффективность на окончательный датасет (подробнее об этом будет рассказано в главе 4).

Теперь, когда вы определились с исходным датасетом, пришло время обратить внимание на модели. Даже если вам кажется заманчивым просто начать разработку пайплайна с нуля, полезно хотя бы ознакомиться с тем, что делают другие.

¹ <https://oreil.ly/tIjl9>

² <http://reddit.com/r/datasets>

³ <https://www.kaggle.com/datasets>

⁴ <https://oreil.ly/BXLA5>

⁵ <https://oreil.ly/Gpv8S>

⁶ <https://commoncrawl.org>

⁷ <https://oreil.ly/kXGiz>

Открытый исходный код

Поиск готового кода может решить две глобальные задачи. Это позволяет, во-первых, увидеть, с какими сложностями столкнулись другие разработчики при выполнении аналогичных задач моделирования, а во-вторых, выявить потенциальные проблемы с имеющимся датасетом. Так что я рекомендую вам поискать и пайплайны для реализации вашего продукта, и код для обработки выбранного вами датасета. Обнаружив пример такого кода, для начала попробуйте самостоятельно получить такие же результаты.

Мне не раз приходилось наблюдать, как после попытки применить найденный в интернете код МО выяснялось, что он не позволяет обучить имеющиеся модели с тем уровнем точности, который он должен обеспечивать по утверждению авторов. Поскольку к описанию новых подходов не всегда прилагается хорошо задокументированный и работающий код, результаты машинного обучения часто трудно повторить, и потому их всегда следует проверять.

Как и в случае поиска данных, при выявлении аналогичной кодовой базы полезно свести задачу к типу входных и выходных данных, а затем поискать кодовые базы для решения задач с аналогичными типами данных.

Например, пытаясь решить задачу генерирования HTML-кода на основе скриншотов веб-страниц, Тони Белтрамелли (Tony Beltramelli), автор статьи «pix2code: генерирование кода на основе скриншотов графического пользовательского интерфейса» («pix2code: Generating Code from a Graphical User Interface Screenshot»¹), понял, что эта задача сводится к транслированию изображения в текстовую последовательность. Это позволило воспользоваться архитектурами и рекомендуемыми практиками из более проработанной области МО — генерации описаний, где текстовая последовательность создается на основе изображения. Таким образом он смог добиться прекрасных результатов при решении совершенно новой задачи и использовать плоды многолетнего труда в смежной сфере применения.

После того как вы найдете данные и код, вы будете готовы двигаться дальше. В идеальном случае это даст вам ориентиры, как лучше начать работу, а также более конкретизированное понимание решаемой задачи. Давайте подытожим, в каких ситуациях вы можете оказаться после поиска готовых решений.

Итоги поиска данных и кода

Как мы только что выяснили, использование уже существующего открытого исходного кода и общедоступных датасетов позволяет ускорить разработку. В худшем случае ни одна из существующих моделей не сможет нормально работать на открытом наборе данных и вы по крайней мере будете знать, что ваш проект потребует значительных усилий по моделированию и/или сбору данных.

¹ <https://oreil.ly/rTQyD>

Если же вам удастся найти готовую модель, предназначенную для решения аналогичной задачи, и обучить ее на датасете, который изначально использовался для ее обучения, останется только адаптировать эту модель к вашей предметной области. При этом я рекомендую придерживаться следующей последовательности действий.

1. Найдите аналогичную модель с открытым исходным кодом, желательно вместе с датасетом, который использовался для ее обучения, и попытайтесь получить такие же результаты обучения.
2. После того как вы воспроизведете результаты обучения, найдите датасет с большей степенью сходства с вашим юзкейсом и попытайтесь обучить полученную ранее модель на этом датасете.
3. После того как вы интегрируете новый датасет с кодом обучения, можно будет оценить производительность модели с помощью выбранных метрик и начать итеративную доработку.

О том, какие сложности может таить каждый из этих этапов и как их можно преодолеть, мы поговорим в части II. А пока что давайте рассмотрим применение только что описанного процесса на нашем учебном примере.

Составление плана разработки МО-редактора

Давайте рассмотрим распространенные рекомендации по написанию текста и поищем датасеты и модели, которые можно было бы применить в нашем МО-редакторе.

Начальный план разработки редактора

Нам следует начать с реализации эвристических алгоритмов. Сделать это можно путем сбора распространенных рекомендаций по написанию и редактированию текстов, например таких, какие были описаны в разделе «Простейший подход: “сам себе алгоритм”» на с. 37.

В идеальном случае наш датасет должен содержать вопросы вместе с соответствующим показателем качества. Прежде всего, мы должны быстро найти легкодоступный аналогичный датасет. В зависимости от производительности, полученной на этом датасете, можно будет выполнить более расширенный или углубленный поиск.

Хорошими примерами текстов с соответствующими метриками качества являются посты в социальных сетях или на онлайн-форумах. Поскольку метрики таких текстов, как правило, служат для выделения наиболее полезного контента, посты часто сопровождаются отметками «нравится».

Широко известным ресурсом для получения ответов на вопросы является сеть веб-сайтов Stack Exchange¹. Весь объем данных сети Stack Exchange сохраняется в обезличенном виде в Архиве интернета², о котором я уже упоминал ранее. Этот датасет прекрасно подойдет в качестве отправной точки.

Мы можем создать начальную модель, используя вопросы сети Stack Exchange, и попытаться предсказать количество отметок «нравится». При этом также можно попробовать разметить этот датасет вручную, чтобы выявить имеющиеся в нем закономерности.

Нам нужно создать модель, способную точно классифицировать качество текста, чтобы затем выдавать рекомендации по написанию текстов. Существует много моделей с открытым исходным кодом, предназначенных для классификации текста; в частности, вы можете ознакомиться с руководством по популярной Python-библиотеке машинного обучения `scikit-learn`³.

После того как у нас будет работоспособный классификатор, можно использовать его для выдачи рекомендаций, чему посвящена глава 7.

Теперь, когда мы определились с начальным датасетом, давайте перейдем к вопросу о том, с какой модели нам лучше начать.

Всегда начинайте с простой модели

Важная идея данной главы состоит в том, что цель создания начальной модели и датасета сводится к получению информативных результатов, способных направить дальнейший процесс моделирования и сбора данных в сторону более полезного продукта.

Начав с простой модели и выделив общие закономерности того, что делает вопрос на сайте Stack Overflow успешным, мы сможем быстро оценить производительность и произвести итеративную доработку.

Противоположный подход, подразумевающий создание с нуля идеальной модели, на практике не работает. Это объясняется тем, что МО представляет собой итеративный процесс. Для быстрого прогресса здесь важно понимать, почему модель может потерпеть неудачу. Чем быстрее вы выясните, где ваша модель может давать сбой, тем быстрее вы продвинетесь вперед. Как выглядит этот итеративный процесс, будет подробно рассказано в части III.

В то же время важно всегда помнить об ограничениях используемого подхода. Например, популярность вопроса зависит не только от качества его формули-

¹ <https://stackexchange.com/>

² <https://oreil.ly/NR6iQ>

³ <https://oreil.ly/y6Qdp>

ровки, но и от множества других факторов. Большую роль играет то, в каком контексте и в каком сообществе публикуется этот пост, насколько популярен автор поста, в какое время дня производится публикация, и много других деталей, которые не принимает в расчет наша исходная модель. Чтобы учесть эти факторы, мы ограничим свой датасет лишь некоторым подмножеством сообществ. Наша первая модель будет игнорировать все метаданные поста, однако мы подумаем о возможности их использования, если это будет необходимо.

Фактически наша модель использует так называемые *слабые метки*, которые слабо коррелируют с целевыми параметрами. После того как мы оценим ее производительность, мы решим, содержат ли эти метки достаточно информации для того, чтобы быть полезными.

Теперь, когда у нас уже есть отправная точка, нужно решить, как добиться прогресса. Устойчивый прогресс в процессе МО часто представляется затруднительным в силу непредсказуемого характера моделирования. Трудно сказать заранее, насколько успешным будет тот или иной метод моделирования. Поэтому хочу поделиться с вами некоторыми советами по обеспечению устойчивого прогресса.

Как обеспечить устойчивый прогресс? Начинайте с простейшего решения

Повторю еще раз: одна из главных сложностей МО, как и в случае разработки любого программного обеспечения, — устоять перед желанием создать ненужные на данный момент элементы. Многие проекты МО терпят неудачу, потому что исходный план сбора данных и создания модели не подвергается регулярной оценке и пересмотру. В силу вероятностного характера машинного обучения часто очень трудно сказать заранее, какой эффект даст использование определенного датасета или модели.

Поэтому *крайне важно* взять в качестве отправной точки самую простую из тех моделей, которые подходят под ваши требования, создать на ее основе сквозной прототип и оценить ее производительность как в плане метрик оптимизации, так и в плане реализации цели продукта.

Начинайте с простого пайплайна

В подавляющем большинстве случаев оценка производительности простой модели на исходном наборе данных является лучшим способом решить, какой задачей следует заниматься далее. На следующих шагах придерживайтесь того же принципа, то есть вводите небольшие и легко отслеживаемые улучшения, вместо того чтобы пытаться сразу создать идеальную модель.

Для этого необходимо создать пайплайн, способный принимать входные данные и возвращать результаты. В большинстве случаев это в действительности подразумевает создание двух отдельных пайплайнов.

Пайплайн обучения

Чтобы ваша модель могла выдавать точные предсказания, сначала ее нужно обучить. Пайплайн обучения принимает на вход все используемые вами размеченные датасеты (которые иногда бывают настолько большими, что не помещаются на одной машине) и передает их модели. Затем он производит обучение модели на датасете до тех пор, пока не будет достигнут удовлетворительный уровень производительности. Обычно с помощью этого пайплайна обучаются несколько моделей, затем их производительность сравнивается на отдельном валидационном датасете.

Пайплайн инференса

Это пайплайн для эксплуатационного окружения, обеспечивающий поставку пользователям результатов обученной модели.

В общих чертах схема работы пайплайна инференса выглядит следующим образом: сначала он принимает входные данные и предобрабатывает их. Обычно этап предобработки состоит из целого ряда операций: очистки и валидации данных, генерирования необходимых для модели признаков и приведения данных к подходящему числовому формату. Пайплайны в более сложных системах также часто требуют дополнительной информации для работы модели, например сведений о пользователях из базы данных. Затем пайплайн прогоняет образец через модель, применяет логику постобработки и возвращает результат.

Типичные схемы пайплайнов обучения и инференса представлены на рис. 2.4. В идеальном случае в обоих пайплайнах должны использоваться одинаковые операции очистки и предобработки, чтобы обученная модель при работе получала данные с тем же форматом и характеристиками.

ОБУЧЕНИЕ



ИНФЕРЕНС



Рис. 2.4. Пайплайны обучения и инференса взаимно дополняют друг друга

Хотя пайплайны для разных моделей могут строиться с учетом разных факторов, описанная общая инфраструктура обычно остается неизменной. Именно поэтому будет очень полезно начать с создания сквозных пайплайнов обучения и инференса и выявить то узкое место, о котором упоминала Моника Рогати в разделе «Моника Рогати: как выбирать и приоритизировать МО-проекты» на с. 40.

Хотя большинство пайплайнов организованы одинаково, они могут сильно различаться по способу выполнения каждого из отдельных этапов в силу различий в структуре датасетов. Давайте проиллюстрируем это на примере пайплайна для нашего редактора.

Пайплайн для МО-редактора

Для нашего редактора мы создадим пайплайны обучения и инференса, используя язык Python — самый популярный язык в сфере МО. При создании первого прототипа наша задача состоит в том, чтобы построить сквозной пайплайн, не слишком беспокоясь о его совершенстве.

Как это обычно бывает при выполнении любой достаточно продолжительной работы, мы можем и *будем* впоследствии возвращаться к отдельным элементам пайплайна, чтобы их улучшить. Для обучения мы будем использовать стандартную разновидность пайплайна, которая широко применяется для многих задач МО и включает в себя следующие основные операции:

- Загрузка записей с данными.
- Очистка данных путем удаления неполных записей и заполнения недостающих значений там, где это необходимо.
- Предобработка и перевод данных в понятный для модели формат.
- Выделение набора данных, который будет использоваться не для обучения, а для валидации результатов модели (валидационного датасета).
- Обучение модели на выделенном подмножестве данных и получение в результате обученной модели и сводной статистики.

Для создания пайплайна инференса мы возьмем часть операций из пайплайна обучения и дополнительно создадим еще несколько операций. В идеальном случае он должен включать в себя следующее:

- Загрузка обученной модели и сохранение ее в памяти (для более быстрого получения результатов).
- Предобработка (такая же, как в случае обучения).
- Сбор любой дополнительной релевантной информации.

- Прогон образца через модель (инференс).
- Постобработка, обеспечивающая очистку результатов перед поставкой их пользователям.

Для большей наглядности часто полезно представить пайплайн в виде блок-схемы, как это сделано на рис. 2.5.

Мы также создадим различные функции исследования и анализа, призванные помочь в диагностике проблем.

- Функция визуализации образцов, на которых модель демонстрирует наилучшие и наихудшие результаты.
- Функция исследования данных.
- Функция изучения результатов модели.

ОБУЧЕНИЕ



ИНФЕРЕНС

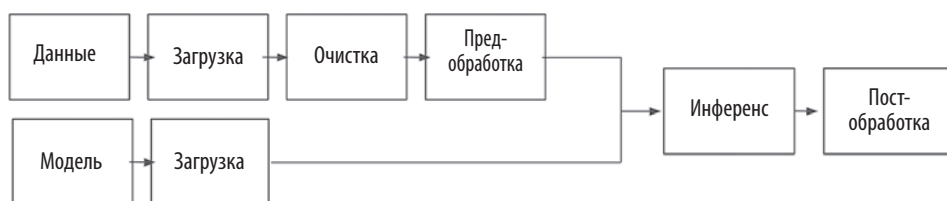


Рис. 2.5. Пайплайны редактора

В состав многих пайплайнов включены этапы валидации входных данных модели и проверки выдаваемых ею результатов. Как вы увидите в главе 10, такие проверки упрощают процесс отладки и позволяют гарантировать определенный уровень качества приложения за счет отсева плохих результатов до отображения их пользователю.

Помните о том, что при использовании МО модель часто выдает непредсказуемые и не всегда приемлемые результаты на незнакомых ей данных. Поэтому важно принять во внимание тот факт, что модель может иногда давать сбой, и учесть вероятность возникновения ошибок в архитектуре системы.

Заключение

В этой главе мы выяснили, как определить основные метрики, позволяющие сравнить совершенно разные модели и оценить плюсы и минусы каждой из них. Мы узнали, с помощью каких ресурсов и методов можно ускорить процесс создания первых пайплайнов. Затем мы в общих чертах рассмотрели, какой набор операций потребуется создать для каждого пайплайна, чтобы получить первый набор результатов.

Теперь, когда у нас сформулирована задача МО, определен способ оценки прогресса и начальный план, пришло время приступить к реализации.

В части II будет более детально описан процесс создания первого пайплайна, а также изучения и визуализации исходного датасета.

ЧАСТЬ II

Создание рабочего пайплайна

Поскольку исследование, обучение и оценка моделей требуют больших затрат времени, выбор неправильного направления МО может обойтись вам дорого. Именно поэтому в книге главное внимание уделяется тому, как снизить риски и расставить приоритеты в работе.

Если в части I мы в основном говорили о том, как спланировать работу, чтобы повысить скорость и шансы на успех, то в данной части вплотную займемся реализацией. Как показано на рис. II.1, при разработке МО-приложений, как и при создании большинства других видов ПО, необходимо как можно быстрее

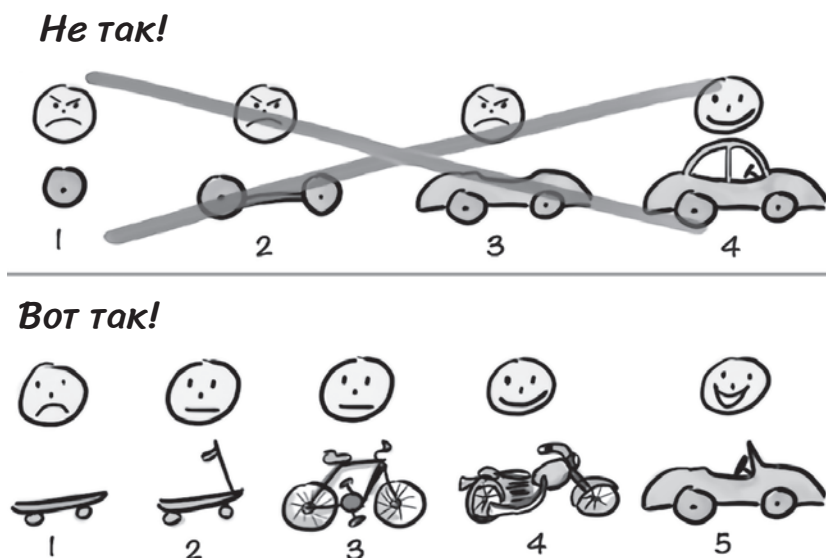


Рис. II.1. Правильный подход к созданию первого пайплайна (рисунок приводится с разрешения Генрика Книберга (Henrick Kniberg))

создать минимально жизнеспособный продукт (minimum viable product, MVP). Именно об этом и пойдет речь: как максимально быстро ввести в действие пайплайн и оценить его эффективность.

Об улучшении модели будет рассказано в части III.

Свою исходную модель мы создадим в два этапа.

Глава 3

В этой главе мы займемся основной структурой нашего приложения: создадим пайплайн для принятия пользовательского ввода и возвращения рекомендаций и пайплайн для обучения моделей перед их использованием.

Глава 4

В этой главе мы сконцентрируемся на сборе и проверке данных для исходного датасета. Задача состоит в том, чтобы быстро выявить закономерности в данных и оценить, какие из них прогнозируемы и могут быть полезны для нашей модели.

ГЛАВА 3

Создание первого сквозного пайплайна

Часть I мы начали с разговора о том, как на основе требований к продукту выбрать подходы к моделированию. Затем мы перешли к этапу планирования и поговорили о том, как искать релевантные ресурсы и использовать их при составлении начального плана разработки. Наконец, мы узнали, что создание исходного прототипа работоспособной системы — лучший способ продвинуться в работе. Именно этому этапу будет посвящена глава 3.

Первая версия приложения не должна быть идеальной. Ее задача — предоставить все элементы пайплайна, чтобы мы могли понять, какие из них следует улучшать в первую очередь. Создание полного прототипа — простейший способ выявить то узкое место, о котором говорила Моника Рогати в разделе «Моника Рогати: как выбирать и приоритизировать МО-проекты» на с. 40.

Давайте начнем с создания простейшего пайплайна для выдачи предсказаний на основе входных данных.

Простейший «каркас» приложения

Мы уже говорили, что большинство МО-моделей включает в себя два пайплайна: пайплайн обучения и пайплайн инференса. Пайплайн обучения позволяет получить высококачественную модель, а пайплайн инференса обеспечивает поставку результатов пользователям. Подробное описание различий между этими пайплайнами см. в разделе «Начинайте с простого пайплайна» на с. 61.

При создании первого прототипа приложения мы сосредоточимся на поставке результатов пользователям. Это означает, что мы начнем разработку с пайплайна инференса. Это позволит нам быстро выяснить, как пользователи могут взаимодействовать с результатами модели, и таким образом собрать полезную информацию, позволяющую упростить обучение модели.

Сосредоточившись на инференсе, на время забудем об обучении. И поскольку пока мы не обучаем модель, мы можем вместо этого составить ряд простых правил. Создание таких правил, или эвристических алгоритмов, часто является прекрасным способом начать работу. Это самый быстрый способ создать прототип и сразу получить упрощенную версию всего приложения.

Хотя это может показаться избыточным, ведь в конечном итоге планируется реализовать решение на базе МО (как это и будет сделано далее в книге), но создание такого прототипа играет важную роль, заставляя нас внимательно изучить задачу и выработать исходный набор гипотез для оптимального решения этой задачи.

Создание, проверка и доработка гипотез наилучшего способа моделирования данных — ключевые составляющие итеративного процесса создания модели, который начинается еще до появления первой модели.



Вот два примера отличных эвристических алгоритмов, примененных в компании Insight Data Science, которые мне довелось курировать.

- *Оценка качества кода.* Решив создать модель, способную на основе примера кода предсказать, какого успеха может добиться программист на сайте HackerRank (посвященном соревновательному программированию), Даниэль начал с подсчета количества открытых и закрытых круглых, квадратных и фигурных скобок.

Поскольку в хорошо работающем коде количество открытых и закрытых скобок, как правило, совпадает, это правило оказалось достаточно эффективной отправной точкой и также позволило понять, что при моделировании следует сосредоточиться на сборе информации о структуре кода с помощью абстрактного синтаксического дерева¹.

- *Подсчет деревьев.* Майк решил создать модель для подсчета количества растущих в городе деревьев на основе спутникового снимка. Изучив некоторые данные, он начал с разработки правила для оценки насыщенности местности деревьями на основе относительной доли зеленых пикселей на изображении.

Как оказалось, этот подход хорошо работает для отдельно стоящих деревьев, но дает сбой, когда приходится иметь дело с большим количеством растущих рядом деревьев. Это позволило понять, что при моделировании следует сосредоточиться на создании пайплайна, способного обрабатывать группы плотно растущих деревьев.

Работу над большинством проектов МО следует начинать с создания такого эвристического алгоритма. Главное — помнить о том, что этот алгоритм нужно создавать на основе экспертных знаний и изучения данных, а затем использо-

¹ <https://oreil.ly/L0ZFk>

вать для подтверждения начальных предположений и ускорения итеративной доработки.

После того как вы разработаете эвристический алгоритм, можно будет создать пайплайн сбора и предобработки данных, применения к ним ваших правил и поставки результатов пользователям. Зачастую для этого достаточно написать запускаемый из консоли Python-скрипт или веб-приложение, принимающее сигнал от камеры пользователя и выдающее результат в реальном времени.

Идея та же, что и в выборе метода МО: максимально упростить продукт, чтобы получить предельно простую работоспособную версию. Создание такого минимально жизнеспособного продукта (MVP) — проверенный способ скорейшего получения полезных результатов.

Прототип МО-редактора

Для нашего МО-редактора возьмем общие рекомендации по редактированию текстов, выработаем ряд правил — что такое хорошие или плохие вопросы, а затем отобразим пользователям результаты применения этих правил.

Чтобы создать минимальную версию нашего продукта, возвращающую рекомендации на основе пользовательского ввода, нам потребуется написать всего четыре функции:

```
input_text = parse_arguments()
processed = clean_input(input_text)
tokenized_sentences = preprocess_input(processed)
suggestions = get_suggestions(tokenized_sentences)
```

Давайте подробно рассмотрим каждую из этих функций. Функция синтаксического анализатора `parse_arguments()` очень простая — она принимает от пользователя строку без каких-либо параметров. Исходный код этого и других примеров кода можно найти в GitHub-репозитории нашей книги.

Парсинг и очистка данных

Прежде всего, мы выполним парсинг данных, принимаемых из командной строки. Это достаточно просто реализовать на языке Python:

```
def parse_arguments():
    """
    :return: Текст, который необходимо отредактировать
    """
```

```
parser = argparse.ArgumentParser(
    description="Receive text to be edited"
)
parser.add_argument(
    'text',
    metavar='input text',
    type=str
)
args = parser.parse_args()
return args.text
```

Перед тем как передавать модели любые входные данные, их сначала нужно валидировать и верифицировать. Поскольку в нашем случае данные вводятся пользователем, мы должны позаботиться о том, чтобы они содержали символы, поддающиеся синтаксическому разбору. Чтобы очистить входные данные, удалим из них любые символы не в кодировке ASCII. Это позволит нам делать обоснованные допущения о содержании текста и в то же время не сильно ограничит креативность наших пользователей.

```
def clean_input(text):
    """
    :param text: Введенный пользователем текст
    :return: Очищенный текст, содержащий только символы в кодировке ASCII
    """
    # В целях упрощения сначала оставим только символы в кодировке ASCII
    return str(text.encode().decode('ascii', errors='ignore'))
```

Теперь нам нужно преобразовать входные данные и выдать рекомендации. Для начала мы можем опереться на результаты исследований в области классификации текста, о которых упоминалось в разделе «Простейший подход: “сам себе алгоритм”» на с. 37. Это подразумевает определение степени сложности текста путем подсчета сводной статистики по количеству слогов, слов и предложений.

Для подсчета статистики на уровне слов мы должны научиться выделять в предложениях отдельные слова. В сфере обработки естественного языка этот процесс называют *токенизацией*.

Токенизация текста

Реализовать токенизацию не так просто, как кажется, поскольку большинство очевидных методов, например разбиение входного текста на слова с помощью пробелов и точек, плохо работает на реальных текстах в силу того, что слова могут отделяться друг от друга множеством разных способов. Например, взгляните

на следующее предложение, которое приводится в качестве примера на курсе по обработке естественного языка¹ в Стэнфордском университете.

«Mr. O'Neill thinks that the boys' stories about Chile's capital aren't amusing.»

Большинство простых методов дадут сбой на этом предложении из-за наличия в нем точек и апострофов с разным смыслом². Вместо того чтобы создавать собственный токенизатор, мы воспользуемся популярной библиотекой с открытым исходным кодом nltk³, которая позволяет реализовать токенизацию двумя простыми действиями:

```
def preprocess_input(text):
    """
    :param text: Очищенный текст
    :return: Текст, подготовленный к анализу: предложения разбиты на лексемы
    """
    sentences = nltk.sent_tokenize(text)
    tokens = [nltk.word_tokenize(sentence) for sentence in sentences]
    return tokens
```

После того как мы преобработаем данные, их можно будет использовать для генерирования признаков, позволяющих оценить качество вопроса.

Генерирование признаков

Последний шаг — написать несколько правил для выдачи рекомендаций пользователям. Для нашего простого прототипа начнем с определения частоты употребления ряда распространенных глаголов, союзов и наречий, а затем вычислим индекс удобочитаемости Флеша (Flesch readability score)⁴. После этого мы можем вывести пользователю отчет с этими метриками:

```
def get_suggestions(sentence_list):
    """
    Возвращает строку, содержащую наши рекомендации
```

¹ <https://oreil.ly/vdrZW>

² Апострофы имеют разное значение, и простое разделение данного предложения на слова по неалфавитным символам не принесет нужного результата. В слове «O'Neill» апостроф отделяет первую букву фамилии; в словах «boys'» и «Chile's» образует притяжательную форму, причем в лексических единицах во множественном числе апостроф стоит после окончания «s»; а в «aren't» он указывает на опускаемую в слове букву. — *Примеч. ред.*

³ <https://www.nltk.org/>

⁴ <https://oreil.ly/iKhmk>

```

:param sentence_list: список из предложений, каждое из которых
является списком слов
:return: рекомендации по улучшению входных данных
"""
told_said_usage = sum(
    (count_word_usage(tokens, ["told", "said"]) for tokens in sentence_list)
)
but_and_usage = sum(
    (count_word_usage(tokens, ["but", "and"]) for tokens in sentence_list)
)
wh_adverbs_usage = sum(
    (
        count_word_usage(
            tokens,
            [
                "when",
                "where",
                "why",
                "whence",
                "whereby",
                "wherein",
                "whereupon",
            ],
        )
        for tokens in sentence_list
    )
)
result_str = ""
adverb_usage = "Adverb usage: %s told/said, %s but/and, %s wh adverbs" % (
    told_said_usage,
    but_and_usage,
    wh_adverbs_usage,
)
result_str += adverb_usage
average_word_length = compute_total_average_word_length(sentence_list)
unique_words_fraction = compute_total_unique_words_fraction(sentence_list)

word_stats = "Average word length %.2f, fraction of unique words %.2f" % (
    average_word_length,
    unique_words_fraction,
)
# Используем HTML-тег для отображения в веб-приложении разрыва строки
result_str += "<br/>"
result_str += word_stats

number_of_syllables = count_total_syllables(sentence_list)
number_of_words = count_total_words(sentence_list)
number_of_sentences = len(sentence_list)

syllable_counts = "%d syllables, %d words, %d sentences" % (
    number_of_syllables,
    number_of_words,
    number_of_sentences,
)

```

```
)
result_str += "<br/>"
result_str += syllable_counts

flesch_score = compute_flesch_reading_ease(
    number_of_syllables, number_of_words, number_of_sentences
)

flesch = "%d syllables, %.2f flesch score: %s" % (
    number_of_syllables,
    flesch_score,
    get_reading_level_from_flesch(flesch_score),
)

result_str += "<br/>"
result_str += flesch

return result_str
```

Вот и всё! Теперь мы можем вызвать свое приложение из командной строки и сразу же увидеть результаты. И хотя пока оно не отличается удобством, мы имеем отправную точку для дальнейшего тестирования и итеративной доработки, чем и займемся далее.

Оцените рабочий процесс

Теперь, имея прототип, можем проверить, насколько правильно мы сформулировали задачу и насколько полезно наше решение. В данном разделе мы поговорим о том, как можно объективно оценить качество исходных правил и выяснить, в удобной ли форме мы предоставляем результаты.

Как сказала ранее Моника Рогати, «часто продукт терпит неудачу, несмотря на успешную работу модели». Если выбранный нами метод будет хорошо справляться с оценкой качества вопросов, но мы не сможем предоставить пользователям какие-либо рекомендации по улучшению формулировки, то практической пользы от нашего продукта не будет, несмотря на высокое качество примененного метода. Давайте рассмотрим весь пайплайн и оценим практическую текущего пользовательского опыта параллельно с результатами нашей вручную сделанной модели.

Пользовательский опыт

Для начала давайте оценим удобство использования продукта безотносительно к качеству модели. Допустим, что в конечном итоге мы получим достаточно эффективную модель. Будет ли в таком случае способ представления результатов пользователям, который используется сейчас, лучшим?

Так, в приложении по подсчету деревьев, вероятно, потребуется представить результаты в виде итогового отчета по всему городу. Нужно указать количество обнаруженных деревьев по городу в целом и по каждому району и измерить ошибку на эталонном тестовом наборе.

Иначе говоря, нужно убедиться в том, что результаты удобны в использовании (или станут таковыми после доработки модели), а также что модель работает хорошо. Именно это мы и сделаем на следующем шаге.

Результаты моделирования

В разделе «Оценка успешности» на с. 44 мы уже говорили о том, как важно сфокусироваться на правильной метрике. Наличие работоспособного прототипа на ранних этапах поможет нам выявить и затем итеративно уточнить набор метрик, отражающий степень успешности продукта.

Так, например, если бы мы решили создать систему, помогающую пользователям искать ближайшие прокатные автомобили, мы могли бы использовать метрику дисконтированного совокупного выигрыша (discounted cumulative gain, DCG). Эта метрика измеряет качество ранжирования: наиболее высокое значение выдается в том случае, когда наиболее релевантные результаты возвращаются раньше других (более подробные сведения о метриках ранжирования см. в статье о DCG¹ в Википедии). При создании первой версии этого продукта мы могли бы допустить, что полезна хотя бы одна из первых пяти рекомендаций, то есть принять DCG равным 5. Однако при тестировании этого продукта на пользователях может оказаться, что они обращают внимание только на первые три результата в выдаче. В таком случае надо изменить метрику успешности на DCG равный 3, а не 5.

Оценивать и пользовательский опыт, и степень эффективности модели нужно, чтобы направить усилия на доработку того, что более важно. Если пользовательский опыт будет плохим, то вам вряд ли поможет улучшение модели. Более того, в таком случае стоит подумать об использовании совершенно другой модели! Давайте рассмотрим два примера.

Нахождение узкого места

Рассмотрев результаты моделирования и текущее представление продукта, мы должны определиться с тем, что делать дальше. Обычно дальше следует итеративная доработка способа представления результатов пользователям (что часто подразумевает изменение метода обучения моделей) или повышение производительности модели путем выявления основных проблемных мест.

¹ https://oreil.ly/b_8Xq

Анализом ошибок мы вплотную займемся в части III, однако уже сейчас следует установить основные виды проблем и возможные способы их устранения. Необходимо определить, что важнее исправить — модель или продукт, поскольку первый и второй случаи требуют разного подхода. Давайте рассмотрим соответствующие примеры.

Проблема с продуктом

Допустим, что вы создали модель, которая должна на основе фотографий научной статьи предсказывать, будет ли она принята к рассмотрению на ведущих конференциях (см. посвященную этому статью Цзя-Биня Хуана (Jia-Bin Huang) «Гештальт научной статьи» («Deep Paper Gestalt»¹)). Однако вы заметили, что простое информирование пользователя о вероятности отказа принесет ему мало пользы. В таком случае вам *не поможет* улучшение модели. Будет разумнее сфокусироваться на извлечении из модели рекомендаций по доработке статьи для повышения шансов на то, что она будет принята к рассмотрению.

Проблема с моделью

Допустим, что вы создали модель для оценки платежеспособности заемщиков и заметили, что она присваивает более высокий риск неплатежеспособности представителям определенной этнической группы при прочих равных факторах. Это может объясняться необъективностью обучающих данных, что можно исправить путем сбора более репрезентативного датасета и создания нового пайплайна его очистки и пополнения. В таком случае вам *потребуется исправить модель*, вне зависимости от способа представления результатов. Поскольку такие ситуации происходят часто, всегда следует учитывать не только агрегатные метрики, но и производительность модели на различных срезах данных. Об этом мы подробно поговорим в главе 5.

Давайте рассмотрим все это на примере нашего МО-редактора.

Оценка прототипа МО-редактора

Давайте посмотрим, насколько хорош наш исходный пайплайн с точки зрения и пользовательского опыта, и производительности модели. Для начала попробуем представить нашему приложению несколько примеров входных данных. Как оно поведет себя в случае простого вопроса, сложного вопроса и целого абзаца?

Поскольку мы используем индекс удобочитаемости, то в идеале надо организовывать рабочий процесс так, чтобы мы получали высокий индекс для простого предложения, низкий индекс для сложного и рекомендации по улучшению для

¹ <https://oreil.ly/RRfIN>

абзаца. Что ж, давайте наконец прогоним эти несколько образцов через наш прототип.

Простой вопрос:

```
$ python ml_editor.py "Is this workflow any good?"
Adverb usage: 0 told/said, 0 but/and, 0 wh adverbs
Average word length 3.67, fraction of unique words 1.00
6 syllables, 5 words, 1 sentences
6 syllables, 100.26 flesch score: Very easy to read
```

Сложный вопрос:

```
$ python ml_editor.py "Here is a needlessly obscure question, that\"
"does not provide clearly which information it would\"
"like to acquire, does it?"
Adverb usage: 0 told/said, 0 but/and, 0 wh adverbs
Average word length 4.86, fraction of unique words 0.90
30 syllables, 18 words, 1 sentences
30 syllables, 47.58 flesch score: Difficult to read
```

Целый абзац (отличный от предыдущих вопросов):

```
$ python ml_editor.py "Ideally, we would like our workflow to return
a positive\"
" score for the simple sentence, a negative score for the convoluted one, and \"
"suggestions for improving our paragraph. Is that the case already?"
Adverb usage: 0 told/said, 1 but/and, 0 wh adverbs
Average word length 4.03, fraction of unique words 0.76
52 syllables, 33 words, 2 sentences
52 syllables, 56.79 flesch score: Fairly difficult to read
```

Давайте посмотрим, возникли ли проблемы с моделью или пользовательским опытом.

Модель

Пока неясно, насколько хорошо наши результаты соответствуют представлениям о качественном тексте. Сложному предложению и целому абзацу был присвоен почти одинаковый индекс удобочитаемости. Признаюсь, мои тексты иногда сложны для восприятия, но все же данный абзац более понятен, чем проверенное перед ним сложное предложение.

Вероятно, извлекаемые атрибуты не всегда хорошо коррелируют с нашими представлениями о «хорошем» тексте. Это объясняется тем, что мы недостаточно четко определили критерий успешности: если мы возьмем два вопроса, как можно определить, какой из них лучше сформулирован? Мы определим это более четко в следующей главе, где займемся созданием датасета.

Очевидно, нам еще потребуется доработать модель, но может быть, мы уже добились удобного представления результатов?

Пользовательский опыт

Исходя из результатов выше, очевидны две проблемы: возвращаемая информация слишком громоздкая и малополезная. Наша цель — предоставить пользователям практичные рекомендации. И хотя признаки и индекс удобочитаемости дают представление о качестве текста, они не помогают пользователю понять, как улучшить свой текст. Вероятно, нам стоит сократить свои рекомендации до одного показателя и добавить практические советы по улучшению текста.

Мы могли бы давать некие общие рекомендации, например использовать меньше наречий, или быть более конкретными, предлагая изменения на уровне слов и предложений. В идеале в выводимом результате могли бы подчеркиваться или выделяться цветом те части входного текста, на которые следует обратить внимание. Макет того, как это может выглядеть, представлен на рис. 3.1.

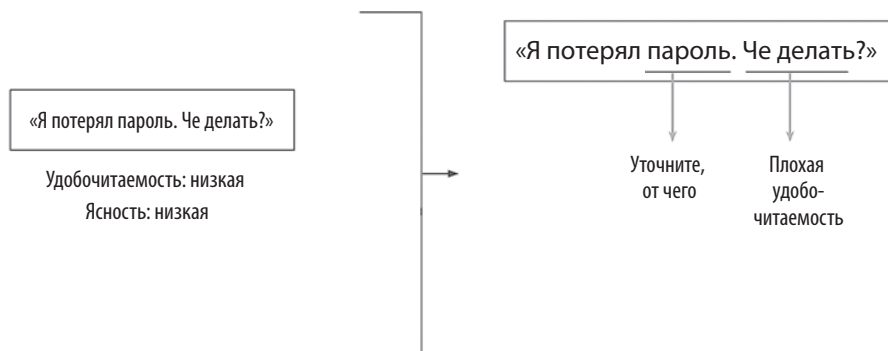


Рис. 3.1. Применимые на практике рекомендации

Даже если мы не сможем давать рекомендации, выделяя отдельные части входной строки, предоставление рекомендаций в формате, показанном справа на рис. 3.1, пойдет нашему продукту на пользу, поскольку такие рекомендации более применимы на практике, чем список показателей.

Заключение

Мы создали исходный прототип инференса и использовали его для оценки качества нашего эвристического алгоритма и последовательности операций.

Это позволило сузить список критериев производительности и итеративно доработать способ представления результатов пользователям.

Мы узнали, что для МО-редактора нам потребуется одновременно и улучшить пользовательский опыт за счет предоставления более практических рекомендаций, и изменить подход к моделированию путем исследования данных для более четкого представления о хорошо сформулированном вопросе.

В первых трех главах мы, исходя из целей продукта, определились с первоначальным подходом, оценили ресурсы и составили план применения подхода. Затем мы создали исходный прототип для проверки составленного плана и сделанных допущений.

Теперь пришло время вплотную заняться тем, что часто упускается из виду в проектах МО, а именно изучением датасета. В главе 4 мы поговорим о том, как собрать исходный датасет, оценить его качество, а затем итеративно разметить подмножества данных, чтобы было проще принимать решения о генерировании признаков и моделировании.

ГЛАВА 4

Получение исходного датасета

Теперь, когда готов план реализации потребностей вашего продукта, а исходный прототип подтверждает работоспособность рабочего процесса и выбранной модели, пришла пора получше изучить датасет. Зачастую тщательное изучение используемых данных позволяет внести наиболее значимый вклад в производительность.

Мы начнем с рассмотрения способов эффективной оценки качества датасета. Затем узнаем, как векторизовать данные и использовать полученное векторизованное представление для более эффективной разметки и исследования датасета. И наконец, разберемся, как на основе проведенного анализа вырабатывать стратегии генерирования признаков.

Давайте начнем с нахождения датасета и оценки его качества.

Итеративная доработка датасета

Лучший способ создания МО-продукта — быстро построить и оценить модель, а затем итеративно дорабатывать ее. При этом важной составляющей успеха моделей являются именно датасеты. Поэтому сбор, подготовка и разметка данных также должны носить *итеративный характер*. Начните с простого датасета, который можно быстро собрать, а затем дорабатывайте его, исходя из результатов исследований.

Такой итеративный подход к данным может поначалу приводить в замешательство. В сфере исследований машинного обучения производительность часто оценивается на стандартных датасетах, которые используются научным сообществом в качестве бенчмарка и потому являются неизменными. А в традиционной разработке ПО мы реализуем детерминированные правила, поэтому данные для нас — это нечто, что нужно получить, обработать и сохранить.

В сфере машинного обучения продукты создаются путем сочетания методов МО и методов разработки ПО. Датасет при этом — один из инструментов для

создания продукта. Выбор исходного датасета, его регулярное обновление и аугментация часто составляют *большую часть работы*. На рис. 4.1 показано, чем отличается рабочий процесс научного МО от промышленного.

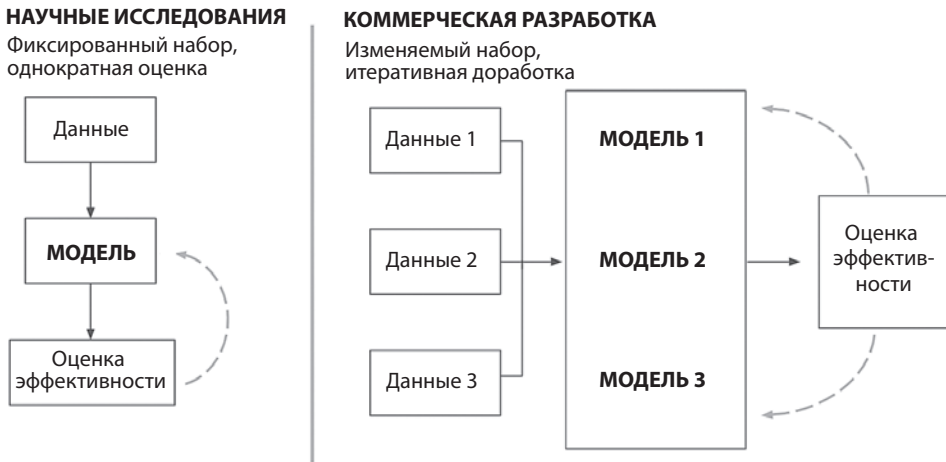


Рис. 4.1. В научных исследованиях датасет фиксирован, а в сфере промышленной разработки является частью продукта

Отношение к данным как к составной части продукта, которую можно (и нужно) дорабатывать, изменять и улучшать, — кардинальная смена парадигмы для новичков в сфере промышленного МО. Но когда вы освоитесь с этим подходом, данные станут для вас первым, к чему вы обратитесь, если что-то пойдет не так.

Проведите анализ данных

Я неоднократно видел, как процесс сбора датасета становился главным препятствием для создания МО-продукта. Одна из причин — тема слабо освещена в обучающих материалах (большинство онлайн-курсов фокусируется на обучении моделей на основе готового датасета). Поэтому многие специалисты боятся этой части работы.

Можно подумать, что работа с данными является просто рутинной, которую необходимо выполнить, прежде чем перейти к более интересной работе с моделями. Однако на самом деле модели являются лишь средством извлечения закономерностей из данных. Таким образом, проверка данных на наличие предсказуемых для модели закономерностей (и на отсутствие смещения) является одной из базовых составляющих работы специалиста по обработке данных (как вы могли заметить, название профессии — не «специалист по моделям»).

В этой главе мы подробно рассмотрим этот процесс, начиная со сбора исходного датасета и заканчивая оценкой его пригодности для МО. Давайте начнем с тщательного изучения датасета, чтобы оценить его качество.

Изучение первого датасета

Так как же исследовать исходный датасет? Конечно, для начала необходимо его собрать. По моим наблюдениям, именно эта задача становится камнем преткновения для многих разработчиков, поскольку они пытаются найти идеальный датасет. Но не забывайте, наша цель — получить простой датасет, позволяющий извлечь некоторые предварительные результаты. Как и многое в МО, это делается по принципу «от простого к сложному».

Начните с малого

Для большинства задач МО увеличение объема данных ведет к повышению качества модели, но это совсем не означает, что вы должны начинать с максимально большого набора данных. Начав работу над проектом с небольшого датасета, вы сможете хорошо изучить свои данные и понять, какую модель лучше для них использовать. Начните с датасета, с которым легко работать, и увеличивайте его размер уже после того, как определитесь с выбором стратегии.

Если ваша компания сохраняет терабайты данных в кластере, можно для начала извлечь однородное подмножество данных, способное уместиться в памяти вашей локальной машины. Но если вы, например, хотите разработать сайд-проект по распознаванию марок автомобилей, проезжающих мимо вашего дома, для начала будет достаточно собрать несколько десятков фотографий движущихся автомобилей.

После того как вы оцените эффективность своей исходной модели и увидите ее слабые места, вы сможете итеративно дорабатывать этот датасет, опираясь на то, что вы узнали. Можно воспользоваться одним из датасетов, предлагаемых в интернете такими платформами, как Kaggle¹ или Reddit², также можно найти несколько примеров самостоятельно. Кроме того, можно собрать данные методом веб-извлечения, используя большие открытые библиотеки данных наподобие тех, что предлагаются на сайте организации Common Crawl³, или генерируя датасет. Подробнее об этом рассказано в разделе «Открытые данные» на с. 55.

¹ <https://www.kaggle.com/>

² <https://www.reddit.com/r/datasets>

³ <https://commoncrawl.org>

Сбор и анализ данных не только необходимы, но и позволяют ускорить процесс работы над проектом, особенно на начальном этапе. Самый простой способ создать хороший пайплайн моделирования и генерирования признаков — посмотреть на свой датасет и изучить его особенности.

Большинство разработчиков придает слишком большое значение работе над моделью и слишком маленькое — работе над данными. В силу этого я всегда рекомендую обратить особое внимание на данные.

Исследуя данные, необходимо проверить наличие закономерностей, но это еще не всё. При создании МО-продукта вы должны задать себе следующие вопросы: «Как использовать эти закономерности в автоматическом режиме?», «Как на основе этих закономерностей создать автоматизированный продукт?».

Инсайт vs продукты

После того как вы получите датасет, можно приступить к изучению его содержимого. При этом важно понимать различие между исследованием данных в аналитических целях и их исследованием с целью разработки продукта. Хотя оба процесса направлены на выявление и изучение закономерностей в данных, в первом случае это делается для того, чтобы извлечь некоторые ценные выводы (например, понять, что большинство незаконных попыток входа на веб-сайт предпринимается во вторник в городе Сиэтл), а во втором случае — чтобы создать новую функциональность на основе выявленных закономерностей (например, сервис, предотвращающий попытки незаконного входа в аккаунты на основе сведений о том, в какое время и с каких IP-адресов предпринимаются такие попытки).

Эта на первый взгляд незначительная разница усложняет создание продукта. Мы должны убедиться в том, что выявленные закономерности будут присутствовать и в тех данных, которые мы будем принимать в дальнейшем, а также количественно оценить различия между данными, используемыми для обучения, и данными, которые мы будем получать от пользователей.

Первый шаг в создании системы предотвращения незаконного входа — выявление сезонного тренда мошеннических попыток входа. Это необходимо, чтобы оценить, насколько часто нам потребуется обучать модели на вновь собранных данных. Далее в главе мы рассмотрим много других примеров по мере более детального изучения данных.

Перед тем как приступать к выявлению тренда, необходимо оценить качество данных. Если выбранный нами датасет не соответствует стандартам качества, мы должны сначала его улучшить, а уже затем приступать к моделированию.

Критерии оценки качества данных

В этом разделе мы рассмотрим, на что следует обращать внимание, начиная работу с новым датасетом. Разные датасеты обладают разной степенью объективности. Необъективность данных требует применения разных инструментов. В силу этого исчерпывающее описание критериев оценки качества датасета выходит за рамки этой книги. В то же время следует отметить ряд критериев, на которые полезно обратить внимание при первом рассмотрении датасета. Давайте начнем с форматирования.

Формат данных

Обратите внимание, отформатирован ли датасет так, что входные и выходные данные не требуют предварительной подготовки и разметки.

Например, обычный датасет для модели, способной предсказать, щелкнет ли пользователь по рекламному объявлению, будет содержать данные по всем щелчкам, выполненным за определенный период времени. Эти данные придется преобразовать так, чтобы они содержали данные о щелчках только по конкретным рекламным объявлениям. Также может понадобиться включить те характеристики пользователя или рекламного объявления, которые, как вам кажется, будут полезными для модели.

Если для вас уже собрали и обработали датасет, нужно убедиться в том, что вы понимаете, как обрабатывались эти данные. Например, если в одном из столбцов датасета хранятся данные по среднему курсу обмена, попробуйте рассчитать этот курс вручную и проверить, совпадают ли полученные значения со значениями в столбце.

В некоторых случаях вы не сможете воспроизвести и проверить этап предобработки. Тогда нужно оценить качество данных и решить, какие признаки следует принять во внимание, а какие лучше проигнорировать.

Качество данных

Перед тем как приступить к моделированию, критически важно оценить качество датасета. Если вы понимаете, что у половины ваших данных отсутствует значение некоторого важного параметра, то вы не потратите часы на отладку модели, пытаясь понять, почему она не работает.

Данные могут быть некачественными в силу самых разных причин: пропусков значений, неточности или даже искажений. Если вы будете иметь четкое представление о качестве данных, это не только позволит вам определиться с приемлемым уровнем производительности, но и упростит задачу выбора подходящих признаков и моделей.

Если вы обрабатываете журналы активности пользователей для прогнозирования трафика веб-продукта, можете ли вы оценить, сколько событий не отражено в журнале? И какая доля сохраненных событий содержит только часть нужной информации о пользователе?

Если вы обрабатываете текст на естественном языке, насколько он качественный? Нет ли в нем, например, непонятных символов или орфографических ошибок?

Если вы обрабатываете изображения, достаточно ли они четкие для того, чтобы решить задачу вручную? Как вам кажется, сможет ли ваша модель распознать объект на изображении, если вы сами с трудом это делаете?

Какая доля ваших данных содержит шум или ошибки? Какая доля входных данных плохо поддается интерпретации и пониманию? Если данные размечены, то всегда ли вы согласны с метками или их точность вызывает у вас сомнения?

Мне как-то довелось работать над проектом по извлечению информации из спутниковых снимков. При разработке такой системы вы в лучшем случае имеете доступ к набору изображений с аннотациями, указывающими расположение интересующих вас объектов, таких как поля или аэродромы. Однако эти аннотации часто могут быть неточными или неполными. Поскольку ошибки такого рода производят значительный эффект при любом подходе к моделированию, об их наличии желательно узнать как можно раньше. Если часть меток отсутствует, мы можем либо вручную разметить датасет, либо найти слабые метки, которые нам подойдут. Однако и первое и второе можно сделать, только *заранее* оценив качество данных.

Помимо проверки формата и качества данных, помочь выявить проблемы может еще один дополнительный шаг: оценка количества данных и распределения признаков.

Количество и распределение данных

Давайте посмотрим, достаточно ли у нас данных и находятся ли значения признаков в допустимых пределах.

Каким объемом данных мы располагаем? Если датасет большой, то нам следует отобрать некоторое его подмножество и провести начальный анализ на нем. А если наш датасет, наоборот, слишком мал или в нем недостает определенных категорий, то обучаемые нами модели могут оказаться столь же необъективными, как и наши данные. Лучший способ избежать такой необъективности — повысить разнообразие данных путем пополнения и аугментации. Хотя качество датасета можно оценивать по-разному, в табл. 4.1 показано, с каких вопросов можно начать.

Таблица 4.1. Критерии оценки качества данных

Качество	Формат	Количество и распределение
Есть ли пропуски значений каких-то полей?	Сколько шагов предобработки требуют ваши данные?	Сколько примеров у вас есть?
Могут ли данные содержать ошибочные значения?	Сможете ли вы проводить такую же предобработку в эксплуатационном окружении?	Сколько примеров приходится на каждую категорию? Все ли категории присутствуют?

Здесь можно привести следующий пример из практики. Специалист по обработке данных, с которым мне довелось работать, хотел создать модель, способную автоматически классифицировать письма, поступающие в службу поддержки, по областям компетенций. У него было девять различных категорий, и на каждую категорию приходился только один пример. Поскольку это слишком малый датасет для обучения модели, он решил сфокусироваться главным образом на стратегии генерирования данных¹. Используя шаблоны общих формулировок для каждой из девяти категорий, он сгенерировал тысячи дополнительных примеров для обучения модели. Эта стратегия позволила ему получить намного более высокий уровень точности пайплайна по сравнению с тем, что могло быть при попытке создать сложную модель, способную обучиться всего лишь на девяти примерах.

Давайте посмотрим, как с помощью такого исследовательского процесса можно оценить качество датасета, выбранного нами для МО-редактора.

Изучение данных для МО-редактора

В качестве датасета для нашего МО-редактора мы изначально решили использовать дампы обезличенных данных сети Stack Exchange². Stack Exchange — это сеть сайтов для получения ответов на вопросы. Вопросы разбиты по темам, например «философия» или «игры». Соответственно, этот дампы данных включает в себя множество архивов отдельных сайтов сети Stack Exchange.

В качестве исходного датасета лучше выбрать архив одного сайта с общими вопросами, чтобы на их основе можно было создать пригодный для использования эвристический алгоритм. По идее, под этот критерий должен хорошо подходить сайт Writing Stack Exchange³, посвященный литературному творчеству.

Архивы отдельных сайтов этой сети предоставляются в виде XML-файла. Нам нужно создать пайплайн, который будет принимать эти файлы и преобразовывать их в текст, из которого затем можно будет извлекать признаки. Вот,

¹ <https://oreil.ly/KRn0B>

² <https://oreil.ly/6jCGY>

³ <https://writing.stackexchange.com/>

например, как выглядит файл `Posts.xml`, содержащий данные сайта `datascience.stackexchange.com`:

```
<?xml version="1.0" encoding="utf-8"?>
<posts>
  <row Id="5" PostTypeId="1" CreationDate="2014-05-13T23:58:30.457"
Score="9" ViewCount="516" Body="&lt;p&gt; &quot;Hello World&quot; example? "
OwnerUserId="5" LastActivityDate="2014-05-14T00:36:31.077"
Title="How can I do simple machine learning without hard-coding behavior?"
Tags="&lt;machine-learning&gt;" AnswerCount="1" CommentCount="1" />
  <row Id="7" PostTypeId="1" AcceptedAnswerId="10" ... />
```

Чтобы использовать эти данные, нам потребуется загрузить XML-файл, преобразовать HTML-теги в текст и представить вопросы и ответы в удобном для анализа формате, например в виде датафрейма (`DataFrame`) библиотеки `pandas`. Именно это и делает представленная ниже функция. Напомню еще раз, что весь используемый в книге код можно найти в `GitHub`-репозитории.

```
import xml.etree.ElementTree as ET
```

```
def parse_xml_to_csv(path, save_path=None):
```

```
    """
```

```
    Открывает дампы постов с расширением .xml, преобразует этот текст в формат CSV
    и одновременно токенизирует его
```

```
    :param path: путь к XML-документу, содержащему посты
```

```
    :return: датафрейм обработанного текста
```

```
    """
```

```
    # Парсинг XML-файла с использованием стандартной библиотеки Python
```

```
    doc = ET.parse(path)
```

```
    root = doc.getroot()
```

```
    # Каждая строка представляет отдельный вопрос
```

```
    all_rows = [row.attrib for row in root.findall("row")]
```

```
    # Отображаем прогресс с помощью модуля tqdm, поскольку
```

```
    # предобработка занимает значительное время
```

```
    for item in tqdm(all_rows):
```

```
        # Извлекаем текст из HTML-кода
```

```
        soup = BeautifulSoup(item["Body"], features="html.parser")
```

```
        item["body_text"] = soup.get_text()
```

```
    # Создаем датафрейм из списка словарей
```

```
    df = pd.DataFrame.from_dict(all_rows)
```

```
    if save_path:
```

```
        df.to_csv(save_path)
```

```
    return df
```

Даже если датафрейм состоит всего из 30 000 вопросов, выполнение процесса занимает больше минуты. Поэтому обработанный файл необходимо сериализо-

вать на диск, чтобы такая обработка выполнялась только один раз. Это можно реализовать с помощью функции `to_csv` библиотеки `pandas`, как показано в последней строке листинга.

Так рекомендуется делать любую необходимую для обучения модели предобработку. Выполнение кода предобработки непосредственно перед оптимизацией модели может существенно замедлить процесс. Поэтому старайтесь по возможности всегда выполнять предобработку данных заранее, сохраняя результаты в csv-формате.

После приведения данных к такому формату мы можем приступить к изучению поставленных ранее вопросов. Полное описание данного исследовательского процесса можно найти в соответствующем ноутбуке в [GitHub-репозитории](#) книги.

Для начала давайте с помощью функции `df.info()` отобразим сводную информацию о структуре нашего датафрейма и посмотрим, нет ли в нем пустых строк. Вот что выводит эта функция:

```
>>>> df.info()

AcceptedAnswerId      4124 non-null float64
AnswerCount           33650 non-null int64
Body                  33650 non-null object
ClosedDate            969 non-null object
CommentCount          33650 non-null int64
CommunityOwnedDate    186 non-null object
CreationDate          33650 non-null object
FavoriteCount          3307 non-null float64
Id                    33650 non-null int64
LastActivityDate       33650 non-null object
LastEditDate          10521 non-null object
LastEditorDisplayName  606 non-null object
LastEditorUserId       9975 non-null float64
OwnerDisplayName       1971 non-null object
OwnerUserId           32117 non-null float64
ParentId              25679 non-null float64
PostTypeId            33650 non-null int64
Score                  33650 non-null int64
Tags                   7971 non-null object
Title                  7971 non-null object
ViewCount              7971 non-null float64
body_text              33650 non-null object
full_text              33650 non-null object
text_len               33650 non-null int64
is_question            33650 non-null bool
```

Как видите, мы имеем чуть больше 31 000 постов, из которых только 4000 с наибольшим получили подходящий ответ. Также можно заметить, что значение признака `Body`, представляющее содержимое поста, иногда является нулевым, что

странно. По идее, все посты должны содержать текст. Изучение строк с нулевым значением `Body` показало, что они относятся к постам, которые не упоминаются в документации датасета, поэтому их можно удалить.

Давайте попробуем наскладку оценить, насколько понятен для нас этот формат. У каждого поста есть значение `PostTypeId`, равное 1 в случае вопроса и 2 в случае ответа. Мы должны узнать, вопросы какого типа получают высокие баллы, и использовать баллы вопросов в качестве слабой метки, замещающей реальную — качество вопроса.

Сначала давайте «состыкуем» вопросы с соответствующими им ответами. Представленный ниже код отбирает все вопросы, получившие ответ, и соединяет их с текстом этого ответа. После этого можно взглянуть на первые несколько строк и убедиться, что ответы действительно соответствуют вопросам, а также визуально оценить качество текста.

```
questions_with_accepted_answers = df[
    df["is_question"] & ~(df["AcceptedAnswerId"].isna())
]
q_and_a = questions_with_accepted_answers.join(
    df[["Text"]], on="AcceptedAnswerId", how="left", rsuffix="_answer"
)

pd.options.display.max_colwidth = 500
q_and_a[["Text", "Text_answer"]][:5]
```

Как видно из табл. 4.2, вопросы и ответы соответствуют друг другу и в текстах практически нет ошибок. Это дает нам уверенность в том, что мы можем должным образом соединить вопросы с ответами.

Таблица 4.2. Вопросы и соответствующие им ответы¹

Id	body_text	body_text_answer
1	«Не()стесненный материально» как пишется: слитно или раздельно? \n	Правильно: нестесненный материально.\nПояснение\pСуществует глагол «стеснить»\nСТЕСНИТЬ 4. кого-что. Лишить свободы действий, стать помехой для кого-, чего-л.; ограничить. Родители не хотели с. сына в выборе невесты. С. в проявлениях свободомыслия. С. в правах. Женильба стеснила его свободу.\nОт глагола можно образовать причастие «не стесненный (чем-либо)», а также прилагательное “нестеснённый” (свободный) с переносным значением.\nk причастию относятся пояснительные слова в форме Т. п., обоз...

¹ Примеры взяты из русского аналога StackExchange — <https://rus.stackexchange.com/questions>. — *Примеч. ред.*

Таблица 4.2 (окончание)

Id	body_text	body_text_answer
2	Как верно употреблять числительные «оба» и «обе» применительно к м. р. + ж. р.? \nНапример: «Мы <i>оба</i> идём к тебе (муж.+жен.)». \n	“Обе”, “обеих” пишется только по отношению к женскому роду. По отношению к мужскому, среднему, общему родам, а также к <i>сочетанию женского рода и другого</i> пишется «оба», «обоих». \n
3	Здравствуйте. Скажите, пожалуйста, после проведения вступительных испытаний, для зачисления абитуриента в вуз необходимо его присутствие в вузе, или можно будет подать заявление о согласии на зачисление онлайн? \nНе могу разобраться со знаками препинания в тексте выше. Подскажите, верно ли? Как расставлять запятые в таких сложных предложениях? \n	В данном случае (в деловом стиле) не все запятые нужны: \nСкажите, пожалуйста, после проведения вступительных испытаний для зачисления абитуриента в вуз необходимо его присутствие в вузе или можно будет подать заявление о согласии на зачисление онлайн? \nВ начале предложения два неоднородных обстоятельства, которые являются общим элементом в сложносочинённом предложении, где простые предложения связаны одиночным союзом ИЛИ. Также общим элементом является вопросительная интонация. \nРозентал...

В качестве дополнительной проверки давайте посмотрим, сколько вопросов не получили ни одного ответа, сколько — как минимум один и сколько ответов было принято.

```
has_accepted_answer = df[df["is_question"] & ~(df["AcceptedAnswerId"].isna())]
no_accepted_answers = df[
    df["is_question"]
    & (df["AcceptedAnswerId"].isna())
    & (df["AnswerCount"] != 0)
]
no_answers = df[
    df["is_question"]
    & (df["AcceptedAnswerId"].isna())
    & (df["AnswerCount"] == 0)
]

print(
    "%s questions with no answers, %s with answers, %s with an accepted answer"
    % (len(no_answers), len(no_accepted_answers), len(has_accepted_answer))
)
```

3584 questions **with** no answers, 5933 **with** answers, 4964 **with** an accepted answer.

Мы имеем здесь относительно равномерное распределение между вопросами, получившими ответ, получившими частичный ответ и не получившими ответа. Это вполне правдоподобный результат для того, чтобы мы могли продолжить свое исследование.

Мы знаем формат данных, и у нас достаточно примеров для начального этапа. Если в ходе работы над своим проектом вы обнаружите, что текущий датасет или слишком мал, или содержит много непонятных признаков, то вам нужно будет собрать дополнительные данные или попробовать применить другой датасет.

Итак, мы располагаем достаточно качественным датасетом и можем двигаться дальше. Пришло время изучить его более глубоко, чтобы обоснованно выбрать стратегии моделирования.

Разметка для выявления трендов в данных

Выявление трендов в датасете не ограничивается лишь оценкой качества. На этом этапе мы должны «встать на место» модели и попытаться предсказать, какую структуру она сможет выявить. Для этого разделим данные на несколько кластеров (этот процесс будет подробно рассмотрен в разделе «Кластеризация» на с. 107) и попытаемся выявить общие черты в каждом из них.

Далее мы пошагово разберем последовательность действий. Сначала сгенерируем сводную статистику нашего датасета, а затем посмотрим, как быстро изучить ее с помощью методов векторизации. Используя векторизацию и кластеризацию, мы проведем эффективное исследование нашего набора данных.

Сводная статистика

В большинстве случаев уместно начать изучение датасета со сводной статистики по всем имеющимся признакам. Это позволит получить общее представление о том, какие признаки представлены в датасете, и найти простой способ выделения категорий.

В МО полезно как можно раньше установить, чем различаются распределения данных разных категорий. Это либо упростит задачу моделирования, либо позволит не переоценить производительность модели, использующей лишь один наиболее информативный признак.

Например, если вы пытаетесь предсказать, какое мнение — положительное или отрицательное — выражают твиты, можно для начала подсчитать, сколько слов в среднем содержит каждый твит. Затем можно построить гистограмму распределения этого признака.

Гистограмма позволит установить, являются ли твиты, выражающие положительное мнение, более короткими. Исходя из этого, можно будет использовать длину твита в словах в качестве прогностического параметра либо собрать дополнительные данные, чтобы выявить еще какую-то информацию.

В качестве иллюстрации давайте попробуем отобразить сводную статистику нашего МО-редактора.

Сводная статистика МО-редактора

Для нашего учебного примера мы можем построить гистограмму распределения длин вопросов, визуально выделив различия между вопросами с высоким и низким баллом. Вот как это можно сделать с помощью библиотеки `pandas`:

```
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle

"""
df содержит вопросы сайта writing.stackexchange.com, вместе с данными
о количестве ответов
Строим две гистограммы:
одну для вопросов, балл которых ниже среднего
одну для вопросов, балл которых выше среднего
В обоих случаях удаляем выбросы, чтобы упростить визуализацию
"""

high_score = df["Score"] > df["Score"].median()
# Отфильтровываем слишком длинные вопросы
normal_length = df["text_len"] < 2000

ax = df[df["is_question"] & high_score & normal_length]["text_len"].hist(
    bins=60,
    density=True,
    histtype="step",
    color="orange",
    linewidth=3,
    grid=False,
    figsize=(16, 10),
)

df[df["is_question"] & ~high_score & normal_length]["text_len"].hist(
    bins=60,
    density=True,
    histtype="step",
    color="purple",
    linewidth=3,
    grid=False,
)

handles = [
    Rectangle((0, 0), 1, 1, color=c, ec="k") for c in ["orange", "purple"]
]
labels = ["High score", "Low score"]
plt.legend(handles, labels)
ax.set_xlabel("Sentence length (characters)")
ax.set_ylabel("Percentage of sentences")
```

Как показывает рис. 4.2, эти распределения не сильно различаются, однако вопросы с высоким баллом, как правило, немного длиннее (что особенно заметно на отметке в 800 символов). Это говорит о том, что важным признаком для модели, предсказывающей балл вопроса, может быть его длина.

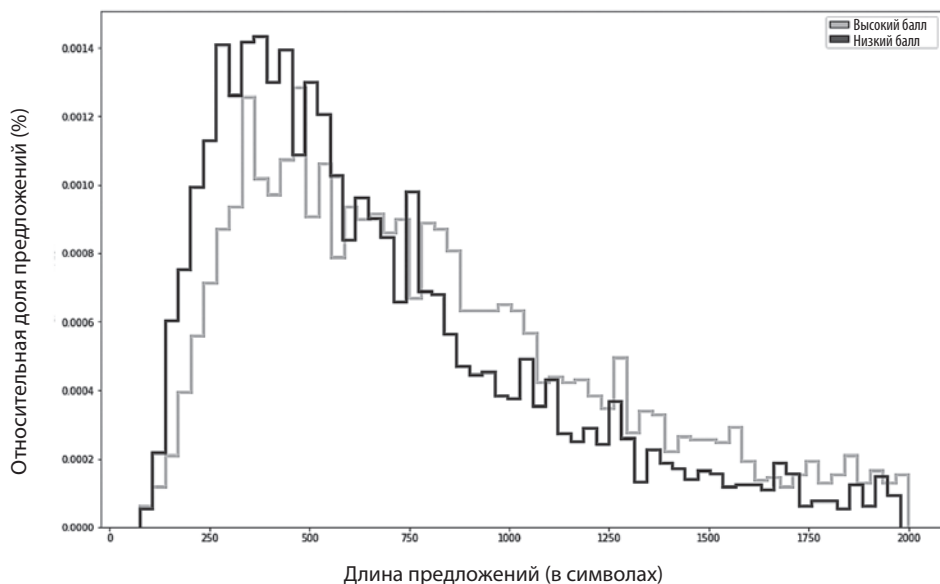


Рис. 4.2. Гистограмма длин вопросов с высоким и низким баллом

Построим аналогичные графики для других признаков, чтобы определить, какие из них потенциально полезны. После этого можно изучить датасет более тщательно, чтобы лучше понимать динамику.

Исследуйте и размечайте эффективно

Пока мы успели изучить только описательную статистику: средние показатели и графики (гистограммы). Чтобы составить четкое представление о данных, необходимо также изучить отдельные элементы. Однако изучение элементов датасета в случайном порядке — не слишком эффективный подход. В этом разделе я покажу вам, как можно добиться максимальной эффективности при визуализации отдельных элементов данных.

Здесь можно воспользоваться таким методом, как *кластеризация*. Под кластеризацией¹ понимается группировка множества объектов таким образом, чтобы объ-

¹ <https://oreil.ly/16f7Z>

екты каждой отдельной группы (называемой *кластером*) имели больше сходства друг с другом, чем с объектами других групп (кластеров). Мы будем использовать кластеризацию не только для изучения данных, но и для анализа предсказаний модели в дальнейшем (см. раздел «Снижение размерности» на с. 106).

Многие алгоритмы кластеризации группируют элементы данных путем измерения расстояния между ними и отнесения близко расположенных элементов к одному кластеру. На рис. 4.3 представлен пример алгоритма кластеризации, разделяющего датасет на три кластера. Кластеризация набора данных — метод обучения без учителя, и часто не существует единственно правильного подхода к его выполнению. В этой книге мы будем использовать кластеризацию для получения определенной структуры, задающей направление наших исследований.



Рис. 4.3. Выделение трех кластеров в наборе данных

Поскольку кластеризация опирается на расстояние между элементами данных, то, какие кластеры мы получим, зависит от выбранного способа численного представления элементов данных. Мы подробно рассмотрим этот вопрос в разделе «Векторизация» на с. 95.

Подавляющее большинство датасетов можно разделить на кластеры на основе их признаков, меток или сочетания признаков и меток. Изучение каждого кластера в отдельности, а также сходств и различий между кластерами является прекрасным способом выявления структуры датасета.

Обратите внимание на следующее:

- Сколько кластеров было выявлено в датасете?
- Имеет ли каждый кластер заметное отличие и в чем именно оно состоит?

- Обладают ли некоторые кластеры существенно более высокой плотностью элементов по сравнению с другими кластерами? Если это так, то ваша модель может плохо работать в областях с меньшей плотностью элементов. Снизить остроту этой проблемы можно путем увеличения числа признаков и объема данных.
- Являются ли данные каждого кластера строго определенными с точки зрения модели? Если элементы данных в каких-либо кластерах отличаются повышенной сложностью, отметьте это для себя, чтобы в будущем, на этапе оценки производительности модели, снова вернуться к ним.

Как уже говорилось выше, алгоритмы кластеризации принимают на вход векторы, поэтому мы не можем передать такому алгоритму просто набор предложений. Чтобы мы могли кластеризовать свои данные, их сначала нужно векторизовать.

Векторизация

Под векторизацией датасета понимается процесс преобразования сырых данных в представляющий их вектор. На рис. 4.4 приведены примеры создания векторных представлений для текста и табличных данных.

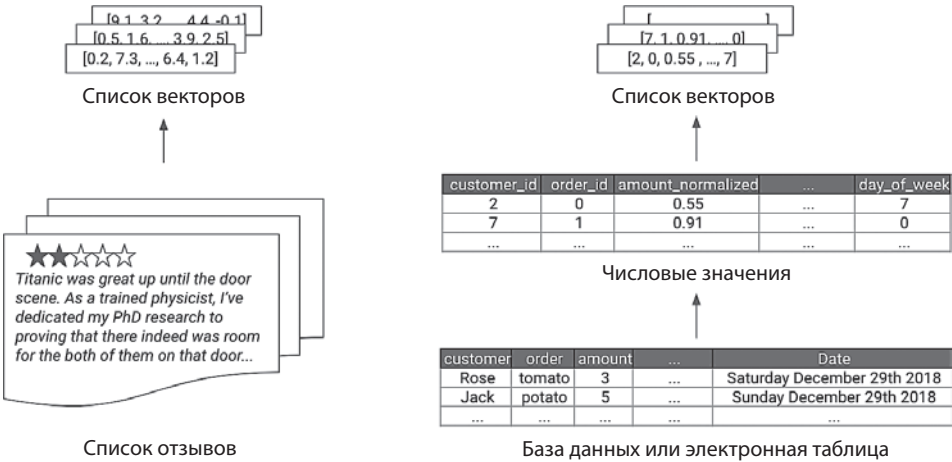


Рис. 4.4. Примеры векторных представлений

Есть много способов векторизации данных. Мы сконцентрируемся на нескольких простых методах, которые работают для наиболее распространенных типов данных: таблиц, текстов и изображений.

Табличные данные. Получить векторное представление для табличных данных, включающих в себя и категориальные, и непрерывные признаки, можно просто путем конкатенации векторных представлений каждого признака.

При этом непрерывные признаки необходимо нормализовать, то есть привести к единому масштабу, чтобы признаки с большим разбросом значений не привели к полному игнорированию моделью признаков с маленьким разбросом. Существуют разные способы нормализации данных, но хорошей отправной точкой будет преобразование каждого признака таким образом, чтобы его среднее значение равнялось нулю, а отклонение — единице. Этот подход часто называют *стандартизированной оценкой*, или *z-оценкой*¹.

Такие категориальные признаки, как, например, цвета, можно преобразовать с помощью унитарного кодирования (*one-hot encoding*). При этом мы получим списки значений признаков, где каждый список состоит из ряда нулей и одной единицы, индекс которой отражает соответствующее значение признака (например, если в датасете четыре цвета, мы могли бы представить красный цвет в виде кода [1, 0, 0, 0] и синий — в виде [0, 0, 1, 0]). Возможно, вас удивляет, почему мы не можем просто присвоить каждому значению признака определенный номер, например, присвоить красному цвету номер 1, а синему цвету номер 3? Так не стоит делать потому, что тогда схема кодировки будет подразумевать некоторую упорядоченность значений (синий цвет будет иметь более высокий номер, чем красный), что часто не соответствует действительности в случае категориальных переменных.

Одна из особенностей *one-hot encoding* состоит в том, что при его использовании расстояние между любыми двумя значениями признака всегда равно единице. Хотя это обеспечивает удобное представление данных для модели, иногда часть значений имеет большую степень сходства друг с другом, чем с остальными данными (так, например, в случае дней недели и суббота и воскресенье являются выходными днями, и в идеале расстояние между их векторами должно быть меньше расстояния между воскресеньем и средой). В последнее время для изучения таких представлений начали с успехом применяться нейронные сети (см. статью «Эмбединги сущностей категориальных переменных» («Entity Embeddings of Categorical Variables»)² Ч. Гуо (C. Guo) и Ф. Берка (F. Berk)). Как оказалось, использование таких представлений вместо других схем кодирования позволяет повысить эффективность моделей.

Наконец, более сложные признаки, например даты, необходимо представить в виде нескольких числовых признаков, отражающих основные характеристики.

Давайте рассмотрим практический пример векторизации табличных данных. Код данного примера можно найти в соответствующем ноутбуке в GitHub-репозитории книги.

Давайте предположим, что вместо того, чтобы рассматривать содержимое вопросов, мы решили предсказывать балл по используемым тегам, количеству

¹ <https://oreil.ly/QTEvI>

² <https://arxiv.org/abs/1604.06737>

комментариев и дате создания. В табл. 4.3 показано, как мог бы выглядеть такой датасет для сайта *writing.stackexchange.com*.

Таблица 4.3. Табличные входные данные без какой-либо обработки

ID	Теги	Количество комментариев	Дата создания	Балл
1	<resources><first-time-author>	7	2010-11-18T20:40:32.857	32
2	<fiction><grammatical-person><third-person>	0	2010-11-18T20:42:31.513	20
3	<publishing><novel><agent>	1	2010-11-18T20:43:28.903	34
5	<plot><short-story><planning><brainstorming>	0	2010-11-18T20:43:59.693	28
7	<fiction><genre><categories>	1	2010-11-18T20:45:44.067	21

Каждый вопрос имеет несколько тегов, дату создания и количество комментариев. Давайте проведем предобработку всех этих данных. Прежде всего, мы должны нормализовать числовые поля:

```
def get_norm(df, col):
    return (df[col] - df[col].mean()) / df[col].std()

tabular_df["NormComment"] = get_norm(tabular_df, "CommentCount")
tabular_df["NormScore"] = get_norm(tabular_df, "Score")
```

Затем необходимо извлечь нужную нам информацию из дат, например год, месяц, день и час публикации поста. Все эти данные представляют собой числовые значения, которые могла бы использовать наша модель.

```
# Преобразуем нашу дату в объект datetime библиотеки pandas
tabular_df["date"] = pd.to_datetime(tabular_df["CreationDate"])

# Извлекаем значимые признаки из объекта datetime
tabular_df["year"] = tabular_df["date"].dt.year
tabular_df["month"] = tabular_df["date"].dt.month
tabular_df["day"] = tabular_df["date"].dt.day
tabular_df["hour"] = tabular_df["date"].dt.hour
```

Наши теги представляют собой категориальные признаки, и каждый вопрос теоретически может иметь произвольное количество тегов. Как мы уже знаем, самый простой способ представления категориальных входных данных состоит в их преобразовании с помощью one-hot encoding. При этом все теги преобразу-

ются в отдельные столбцы, каждый из которых содержит 1 только в том случае, если соответствующий тег используется в вопросе.

Поскольку всего в нашем датасете более трехсот тегов, давайте ограничимся здесь созданием столбцов только для пяти наиболее популярных тегов, которые встречаются более чем в пяти сотнях вопросов. Конечно, мы могли бы взять в расчет каждый тег, однако поскольку большинство из них встречается лишь однажды, это не помогло бы нам в выявлении закономерностей.

```
# Отбираем наши теги, представленные в виде строк, и преобразуем их в массивы
tags = tabular_df["Tags"]
clean_tags = tags.str.split("><").apply(
    lambda x: [a.strip("<").strip(">") for a in x])

# Получаем dummy-значения с помощью функции get_dummies библиотеки pandas
# Отбираем только те теги, которые используются более 500 раз
tag_columns = pd.get_dummies(clean_tags.apply(pd.Series).stack()).sum(level=0)
all_tags = tag_columns.astype(bool).sum(axis=0).sort_values(ascending=False)
top_tags = all_tags[all_tags > 500]
top_tag_columns = tag_columns[top_tags.index]

# Добавляем теги обратно в наш исходный датасет
final = pd.concat([tabular_df, top_tag_columns], axis=1)

# Сохраняем только векторизованные признаки
col_to_keep = ["year", "month", "day", "hour", "NormComment",
               "NormScore"] + list(top_tags.index)
final_features = final[col_to_keep]
```

Как видно из табл. 4.4, теперь наши данные полностью векторизованы и каждая строка содержит только числовые значения. Эти данные уже можно передать алгоритму кластеризации или модели машинного обучения с учителем.

Таблица 4.4. Векторизованные табличные входные данные

ID	Год	Месяц	День	Час	Нормированное количество комментариев	Нормированный балл	Писательское мастерство	Художественная проза	Стиль	Персонажи	Методы	Роман	Публикация
1	2010	11	18	20	0.165706	0.140501	0	0	0	0	0	0	0
2	2010	11	18	20	-0.103524	0.077674	0	1	0	0	0	0	0
3	2010	11	18	20	-0.065063	0.150972	0	0	0	0	0	1	1
5	2010	11	18	20	-0.103524	0.119558	0	0	0	0	0	0	0
7	2010	11	18	20	-0.065063	0.082909	0	1	0	0	0	0	0

ВЕКТОРИЗАЦИЯ И УТЕЧКА ДАННЫХ

Вы будете, как правило, использовать одни и те же методы векторизации и для визуализации данных, и для представления их модели. Однако между первым и вторым есть важное различие. При векторизации данных для модели необходимо векторизовать обучающие данные и сохранить параметры, применявшиеся для получения обучающих векторов. Затем следует применить такие же параметры для валидационного и тестового наборов.

А при проведении нормализации вы должны вычислять сводную статистику, например среднее значение и стандартное отклонение, только для обучающего набора (используя те же значения для нормализации валидационных данных) и для инференса в эксплуатационном окружении.

Нормализация, или выбор категорий для one-hot encoding и валидационных и обучающих данных, может привести к утечке данных из-за использования информации, не входящей в обучающий набор. Это приведет к искусственному завышению производительности модели при обучении и более низкому значению в эксплуатационном окружении. Мы обсудим это подробнее в разделе «Утечка данных» на с. 131.

Различные типы данных требуют применения различных методов векторизации. В частности, для текстовых данных часто нужен более творческий подход.

Текстовые данные. Самый простой способ векторизации текста состоит в использовании вектора подсчета слов, представляющего собой аналог one-hot encoding. Начните с составления словаря уникальных слов вашего датасета. Присвойте каждому слову словаря индекс (в диапазоне от 0 до размера словаря). После этого каждое предложение или абзац можно будет представить в виде списка, длина которого будет равна длине словаря. При этом в значении каждого индекса будет отражаться количество вхождений соответствующего слова в данное предложение.

Этот подход игнорирует порядок слов в предложении и носит соответствующее название «мешок слов» (bag of words). На рис. 4.5 показаны два примера представления предложений в виде мешка слов. Оба этих предложения преобразованы в векторы с информацией о том, сколько раз то или иное слово встречается в предложении, порядок слов при этом не учтен.

Представление текста в виде «мешка слов» и нормализованная версия этого подхода, TF-IDF (Term Frequency — Inverse Document Frequency, частота слова — обратная частота документа), легко реализуются с помощью библиотеки scikit-learn:

```
# Создаем экземпляр векторизатора TFIDF
# Также можно использовать класс CountVectorizer для получения
# ненормализованной версии
```

```
vectorizer = TfidfVectorizer()
```

```
# Обучаем векторизатор на вопросах нашего датасета
```

```
# Возвращается массив векторизованного текста
```

```
bag_of_words = vectorizer.fit_transform(df[df["is_question"]]["Text"])
```

	Входной текст
Предложение 1	"Mary is hungry for apples."
...	...
Предложение 345	"John is happy he is not hungry for apples."



Индекс слова	MARY	IS	HUNGRY	HAPPY	FOR	...	APPLES	NOT	JOHN	HE	SAND
Предложение 1	1	1	1	0	1	...	1	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...
Предложение 345	0	2	1	1	1	...	1	1	1	1	0

Рис. 4.5. Преобразование предложений в векторы «мешка слов»

Со временем было разработано множество новых методов векторизации текста, начиная с появившегося в 2013 году метода Word2Vec (см. статью «Эффективная оценка представлений слов в векторном пространстве» («Efficient Estimation of Word Representations in Vector Space»¹) Миколова (Mikolov) и др.) и заканчивая более поздними методами, такими как FastText² (см. статью «Пакет трюков для эффективной классификации текста» («Bag of Tricks for Efficient Text Classification»³) Жулина (Joulin) и др.). Эти методы обучаются на представлениях и находят сходство понятий лучше, чем метод TF-IDF, за счет того, что учитывают, какие слова часто встречаются в похожих контекстах в больших объемах текста, например в Википедии. Подход основан на гипотезе дистрибутивной семантики: лингвистические единицы со сходным контекстом имеют близкие значения.

А если быть точнее, суть этого подхода сводится к тому, чтобы получить вектор для каждого слова и научить модель предсказывать, какого слова не хватает в предложении, на основе векторов окружающих слов. При этом количество принимаемых в расчет соседних слов называется *размером окна*. На рис. 4.6 по-

¹ <https://oreil.ly/gS-AC>

² <https://fasttext.cc/>

³ <https://arxiv.org/abs/1607.01759>

казана схема такой задачи при размере окна, равном двум. Слева на вход простой модели подаются векторы двух слов, стоящих перед целевым словом, и двух слов, стоящих после него. Затем простая модель и значения векторов оптимизируются так, чтобы результат соответствовал вектору недостающего слова.

Для векторизации слов существует много предобученных моделей с открытым исходным кодом. Векторы, полученные с помощью модели, предобученной на большом объеме текстов (таком как Википедия или архив новостей), часто позволяют модели использовать семантическое значение распространенных слов эффективнее.

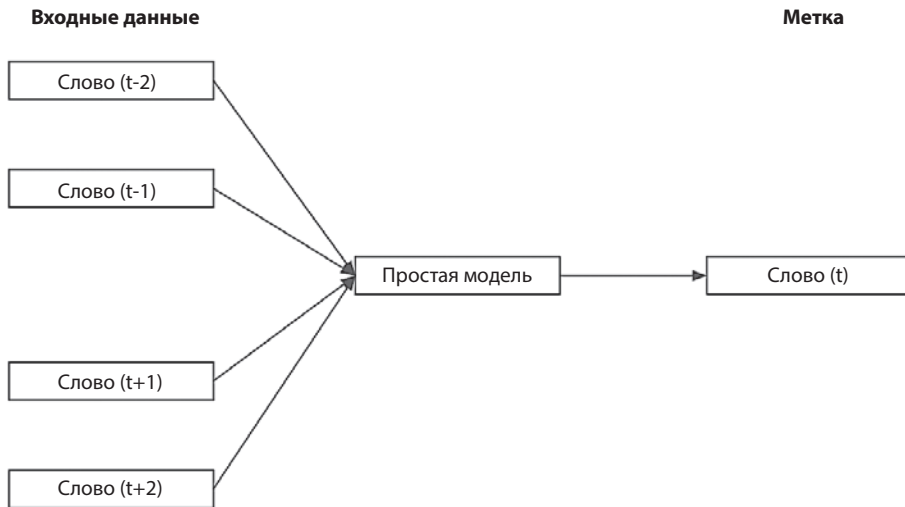


Рис. 4.6. Обучение векторов слов методом Word2Vec с опорой на статью Миколова и др. «Эффективная оценка представлений слов в векторном пространстве»

Например, векторы слов, упомянутые в статье Жулина и др. о методе FastText¹, доступны в виде отдельного онлайн-инструмента. Если вам нужен более настраиваемый метод, вы можете обратиться к SpaCy² — набору инструментов для обработки естественного языка, который предлагает предварительно обученные модели для множества разных задач, а также позволяет легко создать собственную модель.

Ниже показано, как с помощью SpaCy можно загрузить предобученные векторы слов и использовать их для получения семантически значимых векторов предложений. Под капотом SpaCy извлекает предобученные значения для каждого слова нашего датасета (или игнорирует их, если они не использовались на этапе

¹ <https://fasttext.cc/>

² <https://spacy.io>

предобучения), после чего получает представление вопроса, усредняя векторы всех его слов.

```
import spacy
```

```
# Загружаем большую модель и отключаем те части пайплайна, которые не нужны для нашей задачи
# Это значительно ускоряет процесс векторизации
# Подробное описание этой модели см. по ссылке spacy.io/models/en#en_core_web_lg
nlp = spacy.load('en_core_web_lg', disable=["parser", "tagger", "ner",
    "textcat"])
```

```
# Затем просто получаем векторы для каждого из наших вопросов
# По умолчанию возвращается усредненный на основе всех предложений вектор
# Более подробные сведения см. по ссылке https://spacy.io/usage/vectors-similarity
spacy_emb = df[df["is_question"]]["Text"].apply(lambda x: nlp(x).vector)
```

В посвященном векторизации текста ноутбуке GitHub-репозитория книги вы сможете ознакомиться со сравнительным анализом использования для нашего датасета модели TF-IDF и предобученных эмбедингов слов.

С 2018 года очень хорошие результаты демонстрирует векторизация слов с использованием больших языковых моделей даже на более крупных датасетах (см. статью Дж. Говарда (J. Howard) и С. Рудера (S. Ruder) «Универсальная языковая модель тонкой настройки для классификации текста» («Universal Language Model Fine-Tuning for Text Classification»¹) и статью Дж. Девлина (J. Devlin) и др. «BERT: предварительное обучение глубоких двунаправленных преобразователей для понимания языка» («BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding»²)). Однако недостатком этих больших моделей является низкая скорость работы и более высокая сложность по сравнению с простыми эмбедингами слов.

Наконец, давайте поговорим о том, как производится векторизация еще одного распространенного типа данных — изображений.

Визуальные данные. Визуальные данные уже являются векторизованными, поскольку любое изображение представляет собой не что иное, как многомерный массив чисел, или, выражаясь языком машинного обучения, «тензор»³. Так, большинство обычных трехканальных RGB-изображений сохраняется в виде простого списка чисел, длина которого равна высоте изображения в пикселях, умноженной на его ширину и на три (каналы красного, зеленого и синего цвета). На рис. 4.7 показано, как можно представить изображение в виде тензора чисел, отражающих степень интенсивности каждого из трех основных цветов.

¹ <https://arxiv.org/abs/1801.06146>

² <https://arxiv.org/abs/1810.04805>

³ <https://oreil.ly/w7jQi>

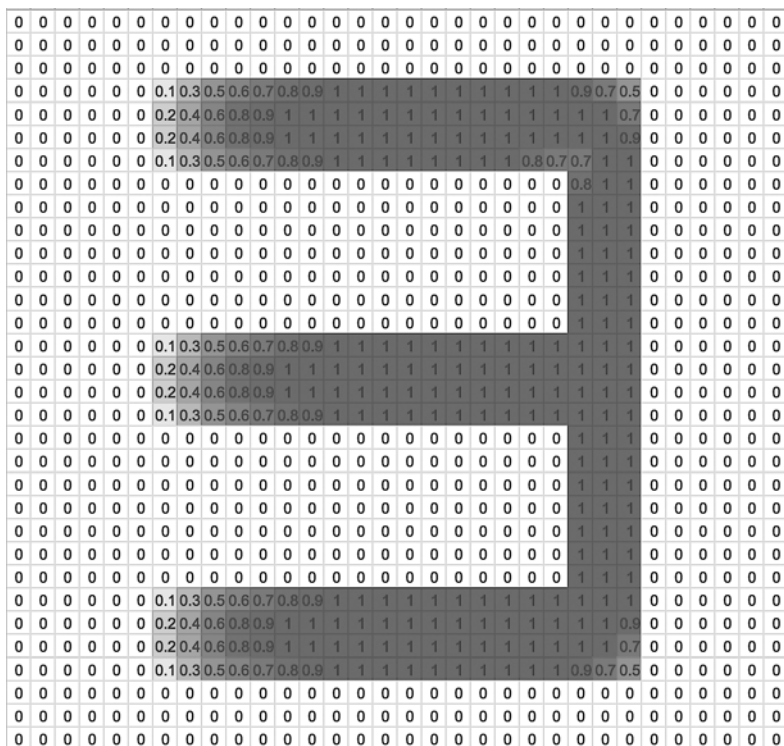


Рис. 4.7. Представление цифры «3» в виде матрицы значений в диапазоне от 0 до 1 (показан только канал красного цвета)

Как выяснилось, извлечение представления на этом слое — чрезвычайно эффективный способ генерирования семантически значимых векторов для изображений. При этом вам не нужно ничего дополнительно настраивать, только загрузить предобученную модель. На рис. 4.8 каждый прямоугольник представляет отдельный слой такой предобученной модели. Выделенное представление является самым полезным. Обычно это представление, которое расположено непосредственно перед классификационным слоем, поскольку оно должно предоставлять наилучшее для классификатора итоговое описание изображения.

Хотя такое представление можно использовать в исходном виде, будет лучше, если наши тензоры будут также отражать содержание изображений. Этого можно добиться, применяя тот же подход, что и для обработки текста, и большие предобученные нейронные сети.

Такие модели, как VGG (см. статью К. Симоняна (A. Simonyan) и А. Циммермана (A. Zimmerman) «Сверхглубокие сверточные сети для крупномасштабного распознавания изображений» («Very Deep Convolutional Networks for Large-Scale Image

Recognition»¹)) или Inception на базе датасета ImageNet² (см. статью К. Шереги (C. Szegedy) и др. «Углубление с использованием сверток» («Going Deeper with Convolutions»³)), обученные на больших датасетах, в итоге выдают очень выразительные представления, способные обеспечить эффективную классификацию. Большинство таких моделей имеют схожую структуру. Поступающее на вход изображение проходит через множество последовательных вычислительных слоев, каждый из которых генерирует собственное представление данного изображения.

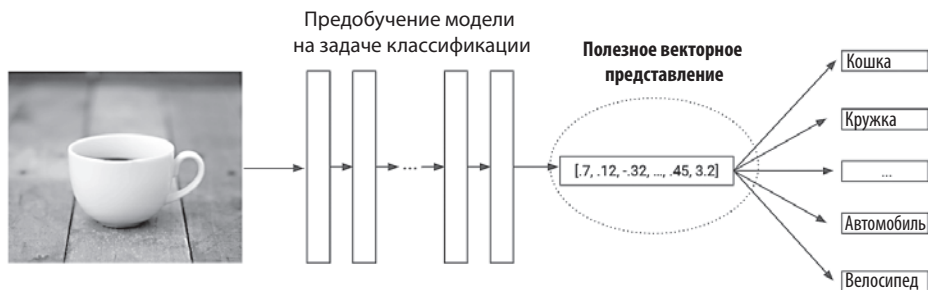


Рис. 4.8. Векторизация изображений с помощью предобученной модели

Наконец, предпоследний слой передается функции, генерирующей классификационные вероятности для каждой категории. Предпоследний слой, таким образом, содержит представление, позволяющее классифицировать объект на изображении. Теперь это представление мы можем использовать для других задач.

Применение таких современных библиотек, как Keras, существенно упрощает выполнение данной задачи. Представленная ниже функция загружает изображения из папки и преобразует их в семантически значимые векторы для последующего анализа, используя для этого сеть библиотеки Keras:

```
import numpy as np

from keras.preprocessing import image
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input

def generate_features(image_paths):
    """
    Принимает массив путей к изображениям
    Возвращает предобученные признаки для каждого изображения
    :param image_paths: массив путей к изображениям
    :return: массив активаторов последнего слоя
```

¹ <https://oreil.ly/TVHID>

² <http://www.image-net.org/>

³ <https://oreil.ly/nbetp>

```

и сопоставление индекса array_index с путем file_path
"""

images = np.zeros(shape=(len(image_paths), 224, 224, 3))

# Загружаем предобученную модель
pretrained_vgg16 = VGG16(weights='imagenet', include_top=True)

# Используем только предпоследний слой
model = Model(inputs=pretrained_vgg16.input,
              outputs=pretrained_vgg16.get_layer('fc2').output)

# Загружаем весь датасет в память (если датасет небольшой)
for i, f in enumerate(image_paths):
    img = image.load_img(f, target_size=(224, 224))
    x_raw = image.img_to_array(img)
    x_expand = np.expand_dims(x_raw, axis=0)
    images[i, :, :, :] = x_expand

# После загрузки всех изображений передаем их модели
inputs = preprocess_input(images)
images_features = model.predict(inputs)
return images_features

```

ПЕРЕНОС ОБУЧЕНИЯ

Предобученные модели используются не только для векторизации данных, их также иногда можно полностью адаптировать под решаемую задачу. Использование предобученной на одном датасете модели для другого датасета или задачи называется переносом обучения (transfer learning). Это не просто повторное использование той же архитектуры или пайплайна, но использование полученных ранее весов обученной модели в качестве отправной точки для решения новой задачи.

Хотя теоретически перенос обучения можно производить между любыми двумя задачами, обычно эта процедура используется для повышения эффективности на небольших датасетах за счет переноса весов с больших датасетов, таких как ImageNet в случае систем машинного зрения и WikiText¹ в случае обработки текста на естественном языке.

Повышая эффективность, перенос обучения также иногда вносит долю нежелательной необъективности. Даже если вы тщательно очистите свой датасет, использование модели, предобученной, скажем, на всем объеме данных Википедии, может привнести присутствующие в этих данных элементы дискриминации по половому признаку (см. статью К. Лу (K. Lu) и др. «Гендерная дискриминация при нейросетевой обработке естественного языка» («Gender Bias in Natural Language Processing»²)).

¹ <https://oreil.ly/voPkP>

² <https://oreil.ly/kPy1l>

После того как вы получите векторные представления данных, их можно кластеризовать или передать непосредственно модели. Их также можно использовать для более эффективного изучения датасета. Группируя элементы со сходными представлениями, вы сможете быстрее выявить тенденции в данных. Давайте поговорим о том, как это сделать.

Снижение размерности

Помимо использования векторных представлений для обучения алгоритмов, их также можно использовать для непосредственной визуализации данных. Это не всегда просто, поскольку получаемые нами векторы часто имеют больше двух измерений, что затрудняет их представление в графическом виде. Например, как можно графически представить 14-мерный вектор?

Джеффри Хинтон (Geoffrey Hinton), удостоенный премии Тьюринга за работу в области глубокого обучения, дает в своей лекции следующий совет: «Чтобы представить, как выглядят гиперплоскости в 14-мерном пространстве, представьте себе трехмерное пространство и громко скажите вслух: “Четырнадцать!”. Это под силу каждому!» (См. слайд 16 из лекции Дж. Хинтона и др. «Обзор основных типов архитектуры нейронных сетей» («An Overview of the Main Types of Neural Network Architecture»¹)). Если вам это кажется сложным, то вы будете рады услышать о таком подходе, как снижение размерности, суть которого состоит в представлении векторов с меньшим количеством измерений при максимальном сохранении их исходной структуры.

Такие методы снижения размерности, как PCA (principal component analysis, метод главных компонент)², t-SNE (см. статью Л. ван дер Маатена (L. van der Maaten) и Дж. Хинтона «Визуализация данных с использованием метода t-SNE» («Visualizing Data Using t-SNE»³)) и UMAP (см. статью Л. Макиннеса (L. McInnes) и др. «UMAP: метод аппроксимации и проецирования равномерных многообразий для снижения размерности» («UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction»⁴)), позволяют проецировать на двумерную плоскость многомерные данные: векторы предложений, изображения или другие признаки.

Эти проекции могут использоваться для выявления в данных закономерностей для их последующего изучения. Однако помните о том, что это приближенные представления реальных данных, и потому любое допущение, принимаемое на основе такого графика, следует проверять с помощью других методов. Так, например, если вы видите, что все кластеры точек, относящихся к определенной

¹ <https://oreil.ly/wORb->

² <https://oreil.ly/kXwvH>

³ <https://oreil.ly/x8S2b>

⁴ <https://oreil.ly/IYrHH>

категории, обладают некоторым общим признаком, убедитесь, что ваша модель действительно использует этот признак.

Для начала представьте свои данные в графическом виде с применением метода снижения размерности, окрасив точки в различные цвета в зависимости от наличия рассматриваемого признака. В случае классификации начните с использования разных цветов для точек с разными метками. В случае обучения без учителя можно окрасить точки в зависимости от значений признаков. Это позволит вам увидеть, какие участки данных будет легче выделить вашей модели, а какие — труднее.

Ниже показано, как это можно легко сделать с помощью метода UMAP, передавая на вход эмбединги, полученные нами в разделе «Векторизация» на с. 95:

```
import umap
```

```
# Обучаем данные методом UMAP и возвращаем преобразованные данные
umap_emb = umap.UMAP().fit_transform(embeddings)

fig = plt.figure(figsize=(16, 10))
color_map = {
    True: '#ff7f0e',
    False: '#1f77b4'
}
plt.scatter(umap_emb[:, 0], umap_emb[:, 1],
            c=[color_map[x] for x in sent_labels],
            s=40, alpha=0.4)
```

Если вы помните, мы решили изначально использовать только данные сайта о литературном творчестве сети Stack Exchange. Результат, полученный для этого датасета, представлен на рис. 4.9. Здесь сразу же можно заметить несколько областей, требующих более внимательного изучения, в частности плотную область не получивших ответа вопросов в верхней левой части графика. Если мы сможем выявить какие-либо общие признаки у этих точек, то получим полезный классификационный признак.

После векторизации и визуализации данных обычно полезно выделить группы со сходными элементами и исследовать их. Это можно делать не только путем простого изучения графика, полученного с помощью метода UMAP, но и путем кластеризации.

Кластеризация

Мы рассматривали ранее кластеризацию как метод выделения структуры в данных. Для чего бы вы ее ни использовали: для изучения датасета или анализа производительности модели (чем мы займемся в главе 5), кластеризация — один из ключевых инструментов в вашем арсенале. Я использую кластеризацию так же, как снижение размерности: в качестве дополнительного способа выявления проблем и интересных элементов данных.

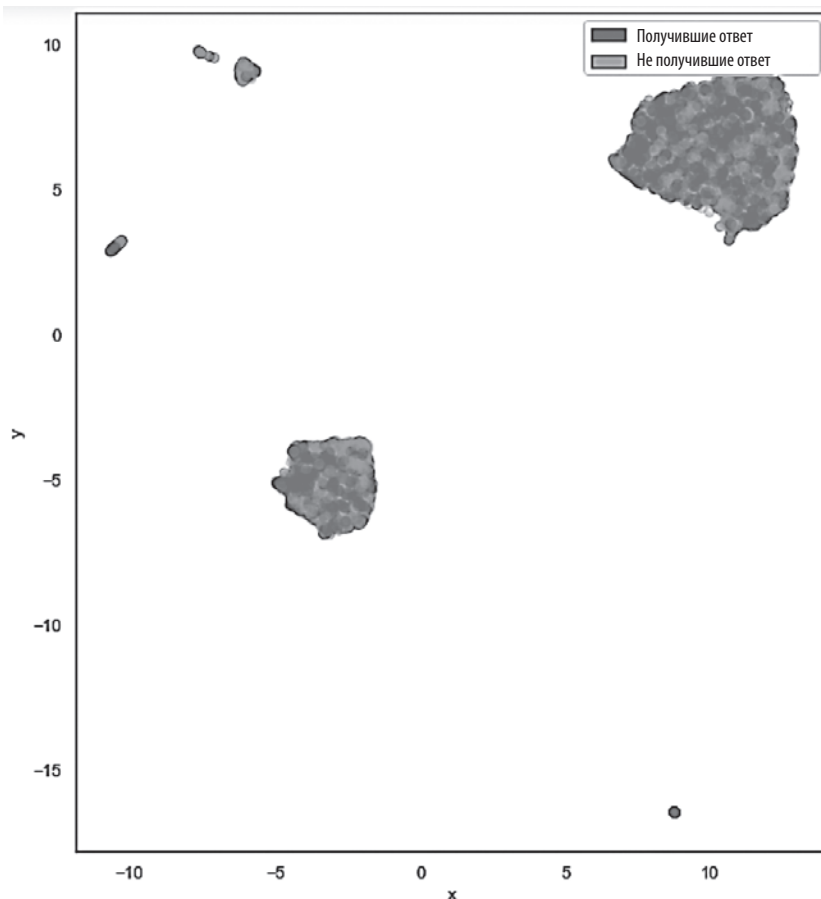


Рис. 4.9. График, полученный с помощью метода UMAP; точки окрашены разным цветом в зависимости от того, получил ли вопрос ответ

На практике простейший способ кластеризации состоит в следующем: попробуйте применить несколько простых алгоритмов, например k -means¹, и настраивайте такие гиперпараметры, как количество кластеров, до тех пор, пока не добьетесь приемлемой производительности.

Степень эффективности кластеризации трудно поддается количественной оценке. На практике для нашего юзкейса будет достаточно визуализировать данные и применить локтевой² или силуэтный метод³. Это не обеспечит идеального

¹ <https://oreil.ly/LKdYP>

² <https://oreil.ly/k98SV>

³ <https://oreil.ly/QGky6>

разделения данных, но позволит выявить те области, где наша модель может испытывать проблемы.

Представленный ниже фрагмент кода демонстрирует, как можно кластеризовать наш датасет, а затем визуализировать полученные кластеры с использованием описанного ранее метода снижения размерности UMAP.

```
from sklearn.cluster import KMeans
import matplotlib.cm as cm

# Выбираем количество кластеров и палитру
n_clusters=3
cmap = plt.get_cmap("Set2")

# обучаем алгоритм кластеризации на наших векторных признаках
clus = KMeans(n_clusters=n_clusters, random_state=10)
clusters = clus.fit_predict(vectorized_features)

# визуализация признаков со сниженной размерностью на 2D-плоскости
plt.scatter(umap_features[:, 0], umap_features[:, 1],
            c=[cmap(x/n_clusters) for x in clusters], s=40, alpha=.4)
plt.title('UMAP projection of questions, colored by clusters', fontsize=14)
```

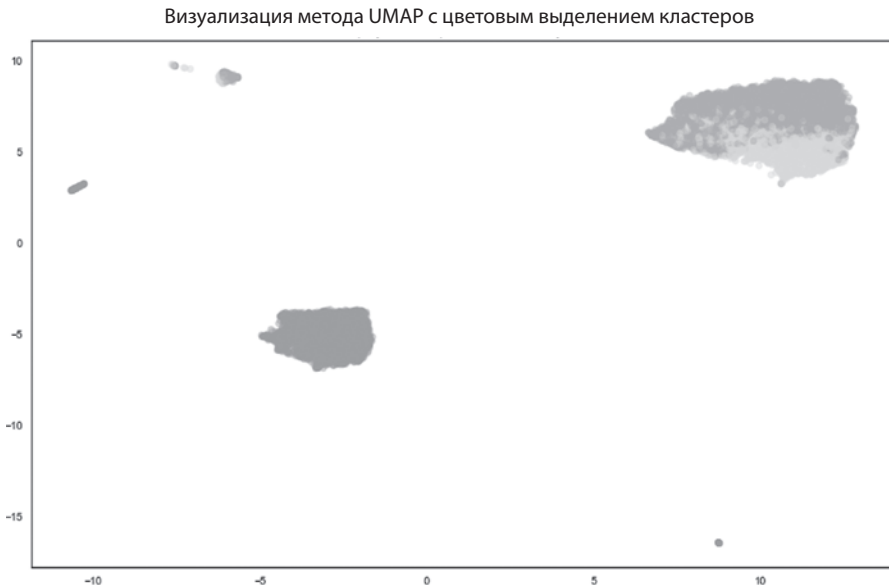


Рис. 4.10. Цветовое распределение кластеров наших вопросов

Как показывает рис. 4.10, кластеры, полученные после обучения на векторных данных, не всегда совпадают с тем, как бы мы разделили эти данные на класте-

ры в двумерном пространстве «на глаз». Причиной этому может быть сложная топология данных или искажения, вносимые алгоритмом сокращения размерности. В принципе, добавление признака кластера часто позволяет повысить эффективность модели, поскольку тем самым мы позволяем ей использовать топологию данных.

После выделения кластеров изучите данные каждого кластера, чтобы выявить имеющиеся в них тренды. Для этого необходимо выбрать несколько точек в каждом кластере и обработать их так, как это сделала бы модель, — пометить правильным, по вашему предположению, ответом. В следующем разделе я расскажу, как лучше провести такую разметку.

Станьте алгоритмом

После изучения агрегатных метрик и кластеризованных данных я рекомендую вам воспользоваться советом из раздела «Моника Рогати: как выбирать и приоритизировать МО-проекты» на с. 40 и попробовать выполнить работу модели: пометить ожидаемым ответом несколько элементов данных в каждом кластере.

Если вы ни разу не попробуете это сделать, то вам будет сложно судить о качестве выдаваемых моделью результатов. Однако если вы попробуете разметить данные вручную, то, возможно, заметите тренды, которые могли бы существенно упростить моделирование.

Если вы помните, тот же совет я давал в разделе об эвристических алгоритмах, так что это не должно быть для вас сюрпризом. При выборе метода моделирования требуется строить не меньше гипотез о данных, чем при создании эвристического алгоритма, и вполне логично, что эти предположения должны быть основаны на данных.

Попробуйте разметить данные даже в том случае, если ваш датасет уже имеет метки. Так вы убедитесь, что метки присвоены правильно. Для нашего учебного примера в качестве слабой метки мы используем полученный вопросом балл. Разметив несколько образцов вручную, мы сможем убедиться в правильности предположения о том, что метка подходит для данного случая.

После того как вы пометите несколько образцов, можно будет скорректировать стратегию векторизации, добавив дополнительно выявленные признаки, чтобы сделать представление данных максимально информативным, а затем продолжить процесс разметки. Схема этого процесса представлена на рис. 4.11.

Чтобы ускорить разметку, проведите предварительный анализ, разметив только несколько элементов данных в каждом выделенном вами кластере и каждое распространенное значение в вашем распределении признаков.



Рис. 4.11. Процесс разметки данных

Один из способов это сделать — применить библиотеки визуализации для интерактивного изучения данных, например библиотеку `Bokeh`¹. Вы можете быстро разметить данные, пройдясь по графику векторизованных образцов и пометив несколько образцов в каждом кластере.

На рис. 4.12 показан репрезентативный образец кластера, содержащего в основном не получившие ответ вопросы. Большинство вопросов этого кластера не получило ответ в силу их расплывчатого характера, затрудняющего выдачу объективного ответа. Таким образом, они верно размечены как «плохие» вопросы. Чтобы ознакомиться с исходным кодом для получения этого графика и примером его использования для разработки МО-редактора, перейдите в ноутбук, посвященный изучению данных для создания признаков, в `GitHub`-репозитории книги.

В ходе разметки можно сохранять метки непосредственно вместе с данными (например, в виде дополнительного столбца датафрейма) или в виде отдельного файла. Это исключительно дело вкуса.

¹ <https://oreil.ly/6eORd>

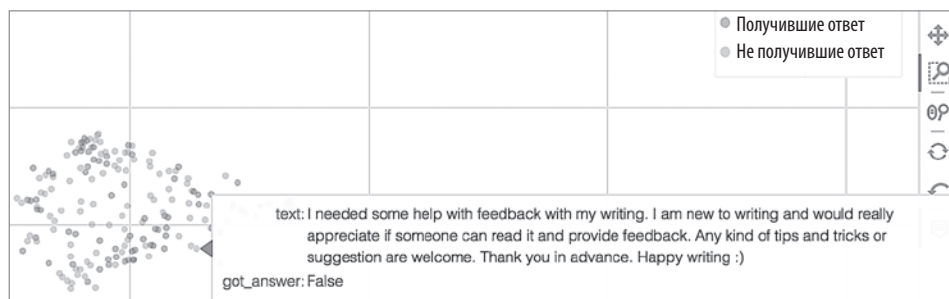


Рис. 4.12. Использование библиотеки Vokeh для изучения и разметки данных

По мере разметки образцов постарайтесь понять, как именно вы делаете тот или иной выбор. Это поможет вам выявить закономерности и сформировать признаки, позволяющие сделать модель более эффективной.

Тренды в данных

Попробовав выполнить разметку, вы в большинстве случаев сможете выявить тренды. Некоторые из них могут быть информативными (например, тот факт, что короткие твиты обычно легче классифицировать как положительные или отрицательные) и позволят вам сформировать полезные для модели признаки. Другие могут быть нерелевантными из-за способа сбора данных.

Например, может случиться так, что все собранные нами твиты на французском языке будут носить отрицательный характер, в результате чего модель будет автоматически классифицировать любые твиты на французском как отрицательные. Как вы думаете, насколько точной будет эта модель при ее использовании на более широкой и репрезентативной выборке?

Однако не стоит отчаиваться в том случае, если вы заметите нечто подобное. Важно лишь выявить такие тренды *до того, как* вы приступите к созданию модели, иначе они приведут к искусственному завышению точности модели на обучающих данных, что, в свою очередь, может привести к ее недостаточной эффективности в эксплуатационном окружении.

Лучший способ борьбы с такими необъективными образцами состоит в том, чтобы собрать дополнительные данные и тем самым сделать обучающий датасет более репрезентативным. Также можно попытаться устранить необъективность путем удаления этих признаков из обучающих данных. Однако это может оказаться неэффективным на практике, поскольку модель может уловить смещение данных, используя корреляции с другими признаками (см. главу 8).

Как только вы определили тренды, пришло время их задействовать. В большинстве случаев это можно сделать двумя способами: либо путем создания признака, отражающего эти тренды, либо путем использования модели, способной легко их использовать.

Используйте данные для принятия решений о признаках и моделях

Выявленные тренды данных следует использовать для принятия обоснованных решений по обработке данных, генерации признаков и стратегии моделирования. Для начала давайте посмотрим, каким образом можно сформировать признаки, способные отразить эти тренды.

Создание признаков на основе закономерностей

МО сводится к использованию статистических алгоритмов для извлечения закономерностей из данных, при этом некоторые закономерности модели уловить проще, чем другие. Возьмем, к примеру, тривиальный случай предсказания числового значения, когда в качестве признака берется само значение, деленное на 2. Чтобы верно предсказывать целевой результат, модели потребуется лишь научиться умножать на 2. С другой стороны, для такой задачи, как предсказание состояния фондового рынка на основе исторических данных, потребуется задействовать намного более сложные закономерности.

Именно поэтому степень практической пользы машинного обучения во многом зависит от успешного создания дополнительных признаков, позволяющих модели выявить полезные закономерности. При этом то, насколько легко дастся модели выявление закономерностей, зависит от способа представления данных и их количества. Чем больше данных у вас имеется и чем меньше шума они содержат, тем меньше придется обрабатывать признаки.

Полезно начать с формирования признаков: во-первых, потому, что в большинстве случаев изначально используется небольшой датасет, а во-вторых, потому, что это позволяет закодировать наши представления о данных и отладить модели.

Широко распространенным примером закономерности, выигрывающей от создания специального признака, является сезонность. Допустим, владельцы интернет-магазина заметили, что большинство покупок совершается в выходные второй половины месяца. В таком случае при разработке модели для прогнозирования последующих продаж им надо убедиться, что модель способна отразить эту закономерность.

Как вы увидите далее, сложность задачи для модели будет зависеть от того, как представлены даты. Большинство моделей может принимать только числовые входные данные (см. описание методов преобразования текста и изображений в числовые входные данные в разделе «Векторизация» на с. 95), поэтому давайте рассмотрим несколько способов представления дат.

Необработанные значения даты и времени

Самый простой способ — использовать Unix-время¹, представляющее собой «количество секунд, прошедших с полуночи (00:00:00) 1 января 1970 года (четверг)».

Несмотря на всю простоту этого способа, модели потребуется выявить довольно сложные закономерности, чтобы научиться определять выходные дни второй половины месяца. Так, последние выходные 2018 года (с 00:00:00 29 декабря до 23:59:59 30 декабря) в формате Unix-времени представляют собой диапазон чисел от 1546041600 до 1546214399. (В этом можно убедиться, вычислив разность между первым и вторым числом: результат будет представлять собой интервал в 23 часа 59 минут и 59 секунд, выраженный в секундах.)

Учитывая то, насколько сложно будет соотнести этот диапазон чисел с выходными днями в других месяцах, можно сделать вывод о том, что при использовании времени в формате Unix модели будет достаточно трудно выделять выходные. Мы можем упростить для нее задачу, прибегнув к извлечению отдельных признаков.

Извлечение дня недели и числа месяца

В частности, мы можем сделать наше представление дат более понятным, выделив день недели и число месяца в два отдельных признака.

При этом, к примеру, время 23:59:59 30 декабря 2018 года будет представлено тем же числом, какое мы использовали раньше, вместе с двумя дополнительными значениями: день недели (0 — воскресенье) и число месяца (30).

Так модели будет проще установить, что повышенная активность покупателей приходится на значения, отражающие выходные дни (0 — воскресенье и 6 — суббота) и вторую половину месяца.

Также следует отметить, что используемый способ представления часто может вносить необъективность. Так, при обозначении дней недели с помощью чисел код пятницы (равный пяти) будет в пять раз больше, чем код понедельника (равный единице). Эта числовая шкала искажает представление, при этом не отражая никакой полезной закономерности.

¹ <https://oreil.ly/hMLX3>

Пересечения признаков

Хотя предыдущий способ представления упрощает задачу для нашей модели, при его использовании все равно придется выявить сложную взаимосвязь между днем недели и числом месяца: высокий трафик приходится только на выходные дни второй половины месяца.

Некоторым видам моделей, в частности глубоким нейронным сетям, удастся улавливать такие взаимосвязи за счет использования нелинейных комбинаций признаков, однако это часто требует большого количества данных. Популярный способ решения этой проблемы состоит в том, чтобы использовать *пересечения признаков*, что еще больше упростит задачу.

Пересечение признаков — это признак, полученный путем простого перемножения двух или более признаков. Введение такой нелинейной комбинации признаков упрощает для модели процесс разграничения данных.

В табл. 4.5 показано, как могут выглядеть элементы данных при использовании описанных выше способов представления.

Таблица 4.5. Представление данных в более понятном виде существенно упрощает для вашего алгоритма задачу получения нужного результата

Человекочитаемое представление	Исходные данные (формат даты и времени в Unix)	День недели	Число	Пересечение (день недели / день месяца)
Saturday, December 29, 2018, 00:00:00	1,546,041,600	7	29	174
Saturday, December 29, 2018, 1:00:00	1,546,045,200	7	29	174
...	...	...	...	...
Sunday, December 30, 2018, 23:59:59	1,546,214,399	1	30	210

На рис. 4.13 видно, как значения признаков изменяются с течением времени и какие из них упрощают для модели задачу выделения определенных элементов данных.

Существует еще один способ представления данных, благодаря которому модели будет еще проще определить прогностическую ценность выходных дней второй половины месяца.

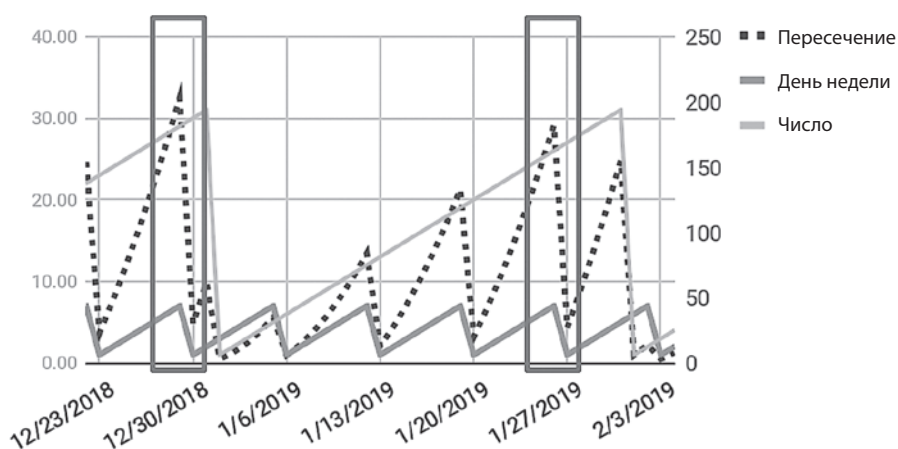


Рис. 4.13. Выделить последние выходные месяца проще всего при использовании пересечений и извлечения отдельных признаков

Предоставление модели ответа

Этот «не совсем честный» способ сводится к следующему. Если вы точно знаете, что определенная комбинация значений признаков обладает особой прогностической ценностью, то вы можете создать новый бинарный признак на их основе. Он будет принимать ненулевое значение при соответствующей комбинации значений. Так, например, в нашем случае можно ввести признак «is_last_two_weekends» (выходные второй половины месяца), который будет равен единице в выходные дни второй половины месяца.

Если окажется, что выходные дни второй половины месяца действительно обладают высокой прогностической ценностью, то модель просто научится использовать этот признак, что сделает ее намного более точной. В ходе разработки МО-продуктов никогда не упускайте возможность упростить стоящую перед моделью задачу. Лучше иметь модель, способную легко решить простую задачу, чем модель, которая не справится со сложной.

Создание признаков — достаточно обширная тема, поскольку разные типы данных требуют разного подхода. В этой книге мы не будем обсуждать, какие признаки лучше создавать для различных типов данных, но если вы хотите ознакомиться с методами и практическими примерами по этой теме, я могу порекомендовать книгу Элис Жэнг (Alice Zheng) и Аманды Казари (Amanda Casari) «Машинное обучение. Конструирование признаков» («*Feature Engineering for Machine Learning*»¹) (O'Reilly).

¹ <http://shop.oreilly.com/product/0636920049081.do>

В общем случае наилучший способ создания полезных признаков сводится к следующему. Изучите свои данные, используя описанные выше методы, и спросите себя: «Как будет проще всего представить данные так, чтобы модель могла выявить в них закономерности?» В следующем разделе я опишу несколько примеров признаков для МО-редактора, созданных мной с помощью этого метода.

Признаки МО-редактора

Изучив датасет для нашего МО-редактора с помощью описанных ранее методов (см. ноутбук, посвященный изучению данных для создания признаков, в GitHub-репозитории этой книги), я создал следующие признаки:

- Прогностической способностью получения ответа на вопрос обладают глаголы действия, такие как *can* («может») и *should* («должен»), поэтому мы добавили бинарный признак, отражающий наличие такого глагола в вопросе.
- Хорошим предиктором также является наличие вопросительного знака, поэтому мы создали соответствующий признак `has_question`.
- Вопросы о правильном использовании английского языка, как правило, не получают ответа, поэтому мы добавили соответствующий признак `is_language_question`.
- Еще одним фактором является длина вопроса. Очень короткие вопросы, как правило, остаются без ответа, поэтому мы добавили нормированный признак длины вопроса.
- В нашем датасете каждый вопрос также имеет заголовок с ключевой информацией, и, как оказалось, изучение заголовка при разметке существенно упрощает задачу. Соответственно, было решено использовать текст как признак.

После определения начального набора признаков можно приступить к созданию модели. О том, как подходить к созданию этой первой модели, мы поговорим далее, в главе 5.

Однако прежде чем переходить к моделям, будет полезно чуть подробнее коснуться вопроса о том, как собирать и обновлять датасет. Для этого я встретился и поговорил с экспертом в данной области Робертом Манро (Robert Munro).

Я надеюсь, что краткое изложение нашей беседы придется вам по душе и вдохновит на чтение следующей части книги, где мы создадим свою первую модель!

Роберт Манро: как находить, размечать и использовать данные

Роберт Манро основал несколько компаний в области ИИ, сформировав передовые команды специалистов в этой области. Он был техническим директором компании Figure Eight, лидирующей в области разметки данных. А до этого Роберт руководил разработкой первого собственного сервиса для обработки естественного языка и машинного перевода в компании AWS. В ходе нашей беседы Роберт поделился рядом уроков, извлеченных им при создании датасетов для МО.

Вопрос. Как вы начинаете работу над проектом МО?

Ответ. Лучше всего начать с изучения бизнес-задачи, поскольку это позволяет задать границы работы. Если говорить о вашем МО-редакторе, то как вы будете предлагать правки: в режиме реального времени во время ввода пользователем текста или будете редактировать готовый текст? Во втором случае можно будет обрабатывать запросы партиями, используя более медленную модель, в то время как в первом случае нужно будет использовать более быструю модель.

Что касается моделей, то первый случай исключает применение sequence-to-sequence моделей, поскольку они работают недостаточно быстро. Кроме того, пока такие алгоритмы не способны на нечто большее, чем рекомендации на уровне предложений, и требуют для обучения больших объемов параллельного текста. Более быстрое решение состоит в том, чтобы задействовать классификатор и использовать в качестве рекомендаций наиболее важные из извлеченных им признаков. Что вам нужно от этой исходной модели, так это простота реализации и уверенность в получаемых результатах. Можно, к примеру, начать с наивного байесовского классификатора на базе «мешка слов».

Наконец, нужно потратить некоторое время на то, чтобы изучить данные и попробовать разметить их вручную. Это даст вам понимание того, насколько сложной является ваша задача и как лучше подходить к ее решению.

Вопрос. Сколько данных вам требуется для того, чтобы начать работу?

Ответ. При сборе данных следует стремиться к тому, чтобы полученный в итоге датасет содержал репрезентативные и разнообразные данные. Для начала изучите датасет и посмотрите, равномерно ли представлены группы, чтобы при необходимости собрать дополнительные данные. Этот процесс часто можно ускорить, кластеризовав датасет и изучив выявленные при этом выбросы.

Что касается разметки данных, то, по опыту, в случае обычной классификации на практике достаточно пометить порядка 1000 образцов самой редкой категории. Количество полученной при этом информации будет как минимум достаточным

для того, чтобы решить, стоит ли продолжать использовать текущий подход к моделированию. После того как будет размечено порядка 10 000 образцов, вы уже можете достаточно уверенно полагаться на используемую модель.

По мере роста количества данных будет постепенно расти и точность вашей модели, что позволит вам получить кривую зависимости степени эффективности от количества данных. При этом вас всегда должна интересовать лишь последняя часть этой кривой, позволяющая судить о том, какую пользу даст дальнейшее увеличение количества данных. Чаще всего разметка большего объема данных дает более значительное улучшение, чем итеративная доработка модели.

Вопрос. Как вы собираете и размечаете данные?

Ответ. Вы можете взять свою лучшую модель и посмотреть, где она «спотыкается». Распространенным подходом здесь является сэмплирование: выявите образцы, на которых ваша модель демонстрирует наибольшую степень неопределенности (то есть расположенные ближе всего к границе, разделяющей группы), и добавьте в обучающий набор аналогичные образцы.

Для этого можно использовать так называемую модель ошибок. Используйте ошибки модели в качестве меток (пометив каждый элемент данных как «предсказанный правильно» или «предсказанный неправильно»). Обучив «модель ошибок» на этих образцах, вы сможете использовать ее на неразмеченных данных, чтобы разметить образцы, на которых, как предсказывает «модель ошибок», основная модель потерпит неудачу.

В качестве альтернативы можно использовать «модель разметки» для поиска тех образцов, которые следует разметить в первую очередь. Допустим, у вас есть миллион образцов, из которых вы успели разметить только 1000. Вы можете создать обучающий набор из 1000 случайно отобранных размеченных изображений и 1000 неразмеченных и научить бинарный классификатор выявлять, какие из них уже размечены. После этого можно будет использовать эту «модель разметки» для выявления и разметки тех элементов данных, которые наиболее сильно отличаются от размеченных.

Вопрос. Как вы понимаете, что обучение моделей принесло определенную пользу?

Ответ. Распространенная ошибка состоит в том, чтобы сосредоточиться при разметке данных лишь на небольшой части датасета. Допустим, что ваша модель дает сбой на статьях, посвященных баскетболу. Если вы начнете размечать дополнительные статьи по теме баскетбола, это может привести к тому, что модель будет хорошо справляться с этой темой, но плохо со всеми остальными. Вот почему даже при использовании некоторой стратегии сбора данных необходимо всегда случайным образом отбирать образцы из тестового набора при валидации модели.

Наконец, следует сказать, что наилучший подход к валидации — постоянно контролировать производительность модели. Вы можете отслеживать неопределенность модели или в идеале привести ее обратно к бизнес-метрикам. Если метрики стали постепенно ухудшаться, это можно объяснять и другими причинами. Будет нелишним провести изучение и, возможно, пополнение вашего обучающего набора.

Заключение

В этой главе мы рассмотрели ряд важных рекомендаций о том, как следует изучать датасет.

Мы начали с оценки качества данных и того, достаточно ли их для достижения нашей цели. Затем мы рассмотрели наилучший способ определения типа данных, который сводится к тому, чтобы сначала посмотреть на сводную статистику, а затем приступить к выявлению общих трендов в кластерах сходных точек.

После этого мы поговорили о том, почему полезно разметить данные и выявить тренды, чтобы на их основе создать полезные признаки. Наконец, мы узнали, какие уроки извлек Роберт Манро, помогая командам специалистов в реализации современных подходов к созданию датасетов для МО.

Теперь, когда мы уже изучили датасет и создали признаки, которые, как мы надеемся, обладают высокой прогностической способностью, пришла пора приступить к созданию нашей первой модели, чем мы и займемся в главе 5.

ЧАСТЬ III

Итеративная доработка моделей

В части I мы говорили о том, как начать работу и отслеживать прогресс МО-проекта. В части II было показано, насколько важно как можно раньше создать сквозной пайплайн и изучить исходный датасет.

В силу своего экспериментального характера МО — это итеративный процесс. Ваш план должен включать постоянную доработку моделей и данных, как это показано в схеме цикла на рис. III.1.

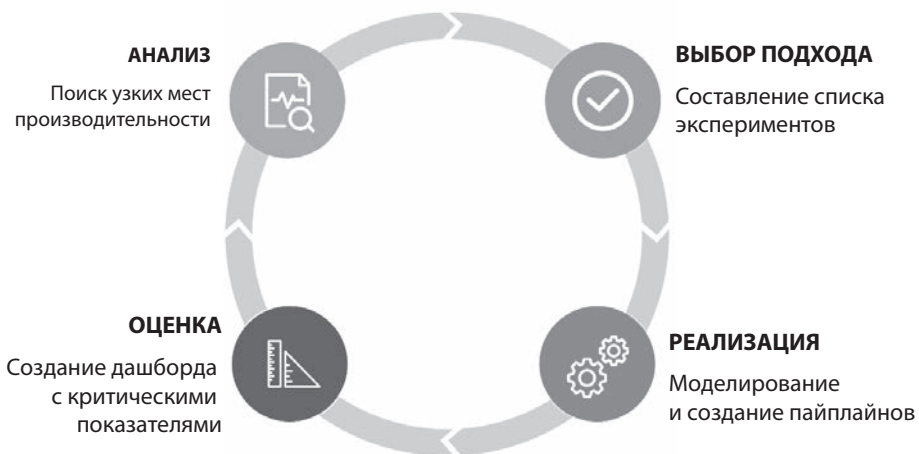


Рис. III.1. Цикл МО

В части III будет описана одна итерация этого цикла. Работая над проектом МО, следует понимать, что для достижения приемлемого уровня эффективности потребуется выполнить много таких итераций. Вот краткий обзор глав этой части книги.

Глава 5

В этой главе мы обучим первую модель, проведем сравнительный анализ производительности и найдем способы ее улучшения.

Глава 6

В этой главе мы поговорим о том, как быстро создать и отладить модель, а также избежать при этом ошибок, отнимающих время.

Глава 7

В этой главе на примере МО-редактора будет показано, как можно использовать обученный классификатор для предоставления пользователям рекомендаций и создания полноценно работающей модели.

Обучение и оценка модели

В предыдущих главах мы рассказали о том, как правильно определить задачу, составить план ее решения, создать простой пайплайн, изучить датасет и сформировать исходный набор признаков. Эти шаги позволяют собрать достаточно информации, чтобы начать обучение адекватной модели. Под «адекватной моделью» здесь понимается модель, которая хорошо подходит для решаемой задачи и с большой вероятностью будет эффективна.

Эту главу мы начнем с краткого обзора тех моментов, на которые следует обратить внимание при выборе модели. Затем рассмотрим способы разделения данных, которые помогут оценить модель в реальных условиях. Наконец, познакомимся с методами анализа результатов моделирования и диагностики ошибок.

Самая простая адекватная модель

Теперь, когда мы уже готовы обучить модель, необходимо решить, с какой модели следует начать. При этом велико искушение опробовать все возможные модели, сравнить их и выбрать ту, которая, согласно метрикам, будет показывать наилучшие результаты на тестовом наборе.

В большинстве случаев это не самый удачный подход. Он не только требует большого объема вычислений (существует множество моделей и параметров, в реальности вы сможете проверить лишь некоторое субоптимальное подмножество), но и рассматривает модели как выдающие предсказания «черные ящики», полностью игнорируя тот факт, что *в способе обучения МО-моделей кодируются неявные предположения о данных*.

Разные модели основываются на разных предположениях о данных и потому предназначены для разных задач. Кроме того, поскольку МО представляет собой итеративный процесс, необходимо выбирать такие модели, которые можно быстро создать и оценить.

Сначала давайте определимся с тем, как выбрать простую модель. После этого рассмотрим несколько примеров закономерностей в данных и подходящих для них моделей.

Простая модель

Простая модель должна быть легкой в реализации, понятной и развертываемой. Легкой в реализации — потому что ваша первая модель, вероятно, не будет последней. Понятной — потому что это упростит для вас процесс отладки модели. Наконец, развертываемой — потому что это одно из основных требований к МО-приложению. Для начала давайте разберемся с тем, что следует понимать под «легкой в реализации» моделью.

Легкая в реализации

Выбранную вами модель должно быть просто реализовать. Обычно это подразумевает, что модель хорошо изучена: для нее написано много обучающих материалов, и вы можете спросить совета у более опытных людей (что особенно актуально в случае выдачи хорошо сформулированных вопросов с помощью нашего МО-редактора). При создании нового приложения на базе МО вам и так придется решить немало проблем с точки зрения обработки данных и выдачи надежного результата; поэтому изначально лучше сделать все возможное, чтобы избежать головной боли, связанной с моделью.

Если это возможно, начните с использования моделей из таких популярных библиотек, как Keras или scikit-learn, и не спешите обращаться к давно не обновлявшимся и плохо документированным экспериментальным GitHub-репозиториям.

После того как вы реализуете модель, вам нужно будет изучить ее и понять, как она использует ваш датасет. Для этого ваша модель должна быть понятной.

Понятная

Под *объяснимостью*, или *интерпретируемостью*, модели понимается способность модели раскрыть причины (например, определенные комбинации предикторов), ведущие к получению тех или иных предсказаний. Хорошая объяснимость может быть полезной в самых разных отношениях. Например, она поможет вам убедиться в отсутствии нежелательного смещения или объяснить разработчику, как можно повысить точность получаемых предсказаний. Она также существенно упростит процесс отладки и итеративной доработки.

Если вам удастся выяснить, на основе каких признаков модель принимает свои решения, это даст вам более четкое понимание того, какие признаки следует до-

бавить, модифицировать или удалить или какая модель могла бы делать более правильный выбор.

К сожалению, даже простые модели часто обладают плохой интерпретируемостью, а более крупные модели иногда вообще не поддаются интерпретации. В разделе «Оценка важности признаков» на с. 152 мы узнаем, как можно решить эту проблему и выявить способы улучшения модели. В частности, мы научимся использовать «интерпретаторы черного ящика» (black box explainers), объясняющие предсказания модели вне зависимости от ее внутреннего устройства.

Такие простые модели, как дерево решений или логистическая регрессия, как правило, легче поддаются объяснению, поскольку предоставляют оценку важности признаков, что является еще одним доводом в пользу их использования в качестве первой модели.

Простая в развертывании

Если вы помните, конечной целью вашей модели является предоставление пользователям некоторого полезного сервиса. Это означает, что при выборе модели всегда следует думать о том, сможете ли вы ее развернуть.

О развертывании мы поговорим в части IV, но уже сейчас попробуем ответить на следующие вопросы:

- Насколько быстро обученная модель будет выдавать предсказание пользователю? При этом нужно учитывать не только время, затрачиваемое моделью на получение предсказания, но и все время ожидания между отправкой пользователем запроса и получением результата. Это включает в себя все операции предобработки (наподобие генерирования признаков), все сетевые вызовы и все операции постобработки: от выдачи предсказания до его представления пользователю.
- Достаточно ли быстрый пайплайн инференса, если учитывать ожидаемое количество одновременно подключенных пользователей?
- Сколько времени занимает процесс обучения модели и как часто нам потребуется ее обучать? Если, например, процесс обучения занимает 12 часов и для поддержания модели в актуальном состоянии ее нужно дообучать каждые 4 часа, то вам придется нести высокие вычислительные затраты, и к тому же ваша модель всегда будет устаревшей.

Мы можем сравнить простоту моделей с помощью таблицы на рис. 5.1. Поскольку по мере развития сферы МО и появления новых инструментов те модели, которые трудно поддаются развертыванию или интерпретации сегодня, могут стать более простыми в использовании, эту таблицу необходимо постоянно обновлять. В силу этого я рекомендую вам создать собственную версию таблицы с учетом вашей предметной области.

Название модели	Простота реализации		Понятность		Развертываемость		Общий «балл за простоту»
	Хорошо изученная модель	Проверенная реализация	Простота извлечения важности признаков	Простота отладки	Длительность инференса	Длительность обучения	
Дерево решений (из библиотеки scikit-learn)	5/5	5/5	4/5	4/5	5/5	5/5	28/30
Сверточная нейронная сеть (СНС) (из библиотеки Keras)	4/5	5/5	3/5	3/5	3/5	2/5	20/30
Преобразователь (из персонального GitHub-репозитория)	2/5	1/5	0/5	0/5	2/5	1/5	6/30

Рис. 5.1. Балльная оценка простоты моделей

Даже после того как вы сузите выбор до простых, понятных и легко развертываемых моделей, у вас еще останется много возможных вариантов. Чтобы выбрать подходящую модель, также потребуется принять во внимание закономерности, выявленные нами в главе 4.

От закономерностей к моделям

Выбор модели должен определяться тем, какие закономерности мы выявили и какие признаки создали. Давайте рассмотрим несколько примеров выбора подходящей модели для имеющихся в данных закономерностей.

Нам нужна независимость от масштаба признаков

Многим моделям труднее использовать признаки с большим разбросом значений, чем с маленьким. В некоторых случаях это хорошо, а в некоторых нежелательно. Для моделей, использующих оптимизацию наподобие градиентного спуска, как это делают нейронные сети, различия в масштабе признаков иногда ведут к нестабильности обучения.

Так, если вы хотите использовать в качестве предикторов и возраст в годах (в диапазоне от 1 до 100), и уровень доходов в долларах (допустим, значения могут достигать девятизначных чисел), то вам нужно проследить за тем, что ваша модель способна использовать наиболее значимые признаки, независимо от их масштаба.

Этого можно добиться путем предварительной нормализации признаков так, чтобы среднее значение равнялось нулю, а стандартное отклонение — единице. Если все признаки будут приведены к общему диапазону, то модель будет учитывать каждый из них в одинаковой мере (во всяком случае, изначально).

Еще одно решение — использовать модели, не подверженные влиянию различий в масштабе признаков. Наиболее распространенными примерами таких моделей являются решающие деревья, случайный лес и градиентный бустинг решающих деревьев. Так, широко используемой в эксплуатационном окружении реализацией градиентного бустинга решающих деревьев является библиотека XGBoost¹, которая показывает хорошие результаты и в плане надежности, и в плане скорости работы.

Предсказываемая переменная является линейной комбинацией предикторов

Иногда есть веские основания полагать, что можно получить достаточно точные предсказания, используя лишь линейную комбинацию имеющихся признаков. В таких случаях следует использовать линейную модель, например линейную регрессию для работы с непрерывными признаками, либо логистическую регрессию, либо наивный байесовский классификатор для задач классификации.

Эти модели отличаются простотой и эффективностью и часто позволяют интерпретировать их веса напрямую, что упрощает выявление важных признаков. Если же вы видите, что отношения между признаками и предсказываемой переменной носят более сложный характер, попробуйте нелинейную модель, например многослойную нейронную сеть либо создание пересечений признаков (см. раздел «Используйте данные для принятия решений о признаках и моделях» на с. 113).

Данные имеют временной аспект

Если вы имеете дело с временным рядом элементов данных, где значение в любой момент времени зависит от предыдущих значений, необходимо использовать модели, которые явным образом кодируют эту информацию. Примерами таких моделей могут служить статистические модели наподобие ARIMA (Autoregressive Integrated Moving Average, авторегрессионное интегрированное скользящее среднее) и рекуррентные нейронные сети (RNN²).

¹ <https://oreil.ly/CWpnk>

² RNN. — *Примеч. ред.*

Каждый элемент данных является комбинацией шаблонов

В частном случае решения задач обработки изображений хорошие результаты показывают сверточные нейронные сети (СНС¹) благодаря тому, что они способны выявлять *трансляционно-инвариантные фильтры*. Это означает, что они выделяют в изображении локальные шаблоны вне зависимости от их относительного положения. Так, научившись обнаруживать глаз, СНС сможет найти его в любой части изображения, а не только в тех местах, где он находился в обучающем наборе.

Сверточные фильтры показывают хорошие результаты и в других областях применения, где приходится иметь дело с локальными шаблонами, например при распознавании речи или классификации текста, где СНС успешно применяются для классификации предложений. Пример реализации такой сети описывает Юн Ким (Yoon Kim) в своей статье «Сверточные нейронные сети для классификации предложений» («Convolutional Neural Networks for Sentence Classification»²).

При выборе правильной модели потребуется учесть еще много других моментов. Для большинства классических задач МО я рекомендую использовать удобную схему³, составленную командой разработчиков библиотеки scikit-learn. В этой схеме можно найти подходящие модели для многих распространенных юзкейсов.

Модель для МО-редактора

В случае МО-редактора нам нужно, чтобы первая модель была быстрой и легко поддавалась отладке. Кроме того, поскольку наши данные представляют собой отдельные образцы, нам не нужно учитывать временной аспект (как, например, в случае ряда последовательных вопросов). Поэтому мы начнем с популярного и безотказного базового варианта — классификатора «случайный лес».

После того как вы выберете подходящую, на ваш взгляд, модель, можно будет приступить к ее обучению. В общем случае не стоит проводить обучение на всем датасете, который вы собрали в главе 4. Лучше сначала обучить модель лишь на некоторой части обучающего набора. Давайте разберемся, почему и каким образом это можно сделать.

Разделение датасета

Основная цель нашей модели — выдача точных предсказаний для данных, введенных пользователем. Это означает, что в конечном итоге модель должна показывать хорошие результаты на *незнакомых ей* данных.

¹ CNN. — *Примеч. ред.*

² <https://arxiv.org/abs/1408.5882>

³ <https://oreil.ly/tUsD6>

Если вы будете использовать один и тот же датасет и для обучения модели, и для оценки ее производительности, то ваша оценка будет отражать лишь степень эффективности модели на уже знакомых ей данных. А если вы будете обучать модель только на некотором подмножестве, то можно будет использовать остальные данные для оценки производительности модели на незнакомых данных.

На рис. 5.2 показано, как можно разделить датасет на три отдельных набора (для обучения, валидации и тестирования) на основе такого атрибута данных, как автор вопроса. Далее в этой главе мы узнаем, что следует понимать под каждым из этих наборов.

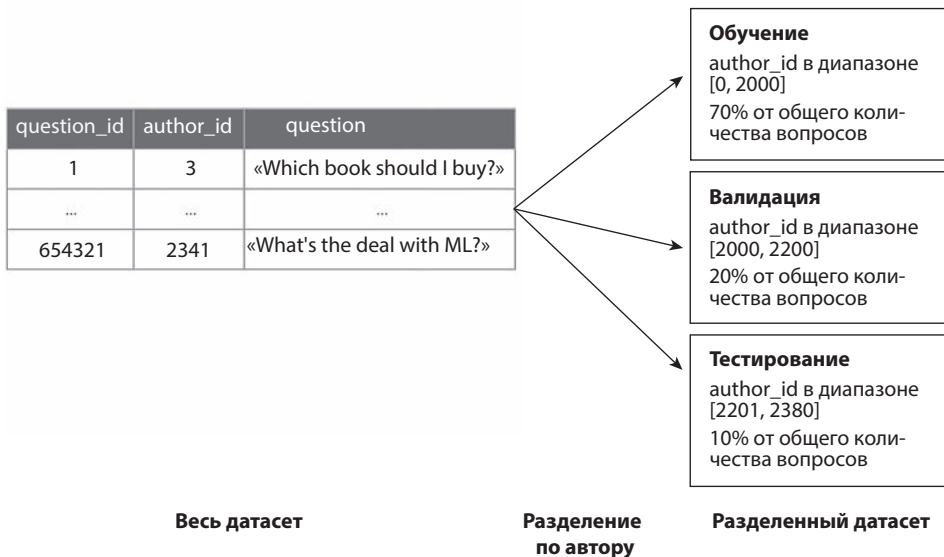


Рис. 5.2. Разделение данных по автору с отнесением определенной доли вопросов к каждому набору

Давайте посмотрим, что собой представляет первый из наборов данных — валидационный.

Валидационный набор

Чтобы оценить производительность модели на незнакомых ей данных, нужно оставить «про запас» часть исходного датасета на этапе обучения, а затем использовать этот резервный датасет для оценки производительности модели в эксплуатационном окружении. Резервный набор позволяет убедиться в том, что модель способна обобщать незнакомые ей данные.

Вы можете отобрать образцы из разных участков датасета для валидационного набора, чтобы оценить модель и провести обучение на остальных данных. Многократное выполнение этого процесса позволяет держать под контролем разброс, обусловленный конкретным содержанием валидационного набора, и называется *кросс-валидацией*.

По мере того как вы будете изменять свою стратегию предобработки данных, тип используемой модели и ее гиперпараметры, будет изменяться и производительность модели на валидационном наборе (в идеальном случае в лучшую сторону). Использование валидационного набора позволяет настраивать гиперпараметры подобно тому, как использование обучающего набора позволяет настраивать параметры модели.

Многократное использование валидационного набора для корректировки модели может привести к тому, что ваш пайплайн моделирования станет «заточенным» под конкретный набор валидационных данных. Это идет вразрез с назначением валидационного набора, который должен представлять собой образец незнакомых данных. Поэтому необходимо зарезервировать дополнительный тестовый набор.

Тестовый набор

Поскольку мы будем выполнять несколько циклов итераций нашей модели, каждый раз оценивая производительность на определенном валидационном наборе, модель может подстроиться под эту выборку. Это поможет модели обобщать данные за пределами обучающего набора, однако может привести к тому, что она будет хорошо работать только на конкретном валидационном наборе. В идеале нам нужно получить модель, которая будет показывать хороший результат на новых, не содержащихся в валидационном наборе данных.

В силу этого необходимо выделить третий набор, называемый *тестовым*, для окончательной оценки производительности модели на незнакомых данных. Использование тестового набора является рекомендуемой практикой, однако иногда разработчики используют вместо него валидационный набор. Это повышает риск «подстройки» модели под валидационный набор, но может подойти, если будет проведено лишь несколько проверок.

Оценку производительности модели на тестовом наборе не следует использовать в качестве основания для принятия решений по моделированию, поскольку этот набор должен представлять собой образец незнакомых данных, с которыми мы будем иметь дело в эксплуатационном окружении. Адаптация метода моделирования под тестовый набор может привести к завышению производительности модели.

Чтобы полученная в итоге модель хорошо работала в эксплуатационном окружении, обучающие данные должны быть похожи на те данные, которые будут

создавать пользователи вашего продукта. В идеале в вашем датасете должны быть представлены абсолютно все разновидности данных, которые вы можете получить от пользователей. Если так не получится, имейте в виду, что производительность на тестовом наборе будет отражать лишь степень эффективности на некотором подмножестве пользователей.

В случае МО-редактора это означает, что наши рекомендации не всегда подойдут тем пользователям, которые не соответствуют демографическим характеристикам аудитории сайта *writing.stackexchange.com*. Если мы хотим решить эту проблему, нужно расширить датасет, чтобы в нем было больше вопросов, созданных другими пользователями. Для начала можно было бы добавить в наш датасет вопросы с других ресурсов сети Stack Exchange или прочих сайтов с ответами на вопросы, тем самым охватив более широкий круг тем.

Такая корректировка датасета не всегда возможна для небольшого проекта. Однако при создании продукта, рассчитанного на массовое использование, необходимо сделать все возможное, чтобы недостатки модели были выявлены до того, как их заметят пользователи. Использование более репрезентативного датасета позволяет исключить значительную часть тех ошибок, которые будут рассмотрены в главе 8.

Относительные пропорции

В общем случае необходимо максимизировать объем обучающих данных, в то же время выделив достаточно большое количество валидационных и тестовых данных для получения точных показателей производительности. На практике часто используются следующие пропорции: 70% данных для обучения, 20% — для валидации и 10% — для тестирования, однако все зависит от размеров общего датасета. Если датасет большой, можно увеличить пропорцию обучающих данных без риска получить слишком маленькие валидационный и тестовый наборы. А если датасет небольшой, приходится уменьшить пропорцию обучающих данных, чтобы количество валидационных и тестовых данных было достаточным для точной оценки производительности.

Теперь, когда мы знаем, зачем следует разделять датасет и что собой представляет каждая из получаемых выборок, встает следующий вопрос: как относить элементы данных к той или иной выборке? Метод разделения оказывает значительное влияние на эффективность моделирования и должен выбираться с учетом конкретных особенностей датасета.

Утечка данных

Процесс валидации сильно зависит от метода разделения данных. При этом необходимо стремиться к тому, чтобы данные валидационного и тестового наборов имели как можно больше сходства с данными, с которыми модель будет работать в дальнейшем.

Разделение данных на обучающий, валидационный и тестовый наборы чаще всего производится путем случайного отбора элементов датасета. В некоторых случаях это может приводить к *утечке данных*. При утечке данных модель получает на этапе обучения информацию, к которой у нее не будет доступа при взаимодействии с реальными пользователями.

Вы должны сделать все возможное, чтобы не допустить утечки данных, поскольку она ведет к завышению производительности модели. При обучении модели на имеющем утечку датасете она может использовать для выдачи предсказаний информацию, которой у нее не будет, когда она столкнется с другими данными. Это искусственно упрощает задачу для модели, но исключительно за счет «просочившейся» информации. Модель покажет высокую эффективность на выделенных данных, но будет работать гораздо хуже при эксплуатации.

На рис. 5.3 представлено несколько распространенных ситуаций, когда разделение данных случайным образом ведет к утечке. Она может быть вызвана множеством разных причин, и далее мы изучим два наиболее частых случая.



Рис. 5.3. Случайное разделение данных часто ведет к утечке

Сначала рассмотрим утечку данных временных рядов (верхний пример на рис. 5.3). Затем проанализируем случай засорения выборки (два нижних примера на рис. 5.3).

Утечка данных временных рядов. В случае прогнозирования временных рядов модель должна учиться на данных за прошедший период, чтобы предсказывать события, которые еще не произошли. Если мы случайным образом разделим датасет прогнозирующей модели, это приведет к утечке данных, поскольку модель, обученная на случайном наборе точек и проверенная на остальных данных, будет иметь доступ к обучающим данным, расположенным во времени *после* предсказываемых ею событий.

Модель будет показывать завышенную производительность на валидационном и тестовом наборах, но окажется неэффективной в эксплуатации, поскольку в действительности она просто научится использовать информацию о будущих событиях, которая будет недоступна в реальных условиях.

Зная о существовании такой проблемы, можно в большинстве случаев легко избавиться от утечки временных данных. При других видах утечки данных модель может получить доступ к информации, которая должна быть ей недоступна на этапе обучения, что ведет к искусственному завышению производительности за счет «засорения» обучающих данных. Многие из этих видов утечки данных гораздо труднее поддаются выявлению.

Засорение выборки. Утечка данных часто происходит на том уровне, где данные разделяются случайным образом. Как-то мне довелось помогать одному специалисту по обработке данных в создании модели, предсказывающей оценки учеников за сочинение. Созданная им модель показала почти идеальные результаты на тестовом наборе.

Когда модель демонстрирует очень хорошие результаты в решении настолько сложной задачи, ее следует тщательно изучить, поскольку это часто объясняется наличием ошибки или *утечки данных*. Здесь справедлив аналог закона Мерфи в сфере МО: чем больше вас удивляет высокая производительность модели на тестовых данных, тем больше вероятность того, что ваш пайплайн содержит ошибку.

В описываемом мной случае произошло следующее. Поскольку почти все ученики написали по несколько сочинений, разделение данных путем случайного отбора привело к тому, что сочинения одного и того же ученика могли присутствовать и в обучающем, и в тестовом наборе. Это позволило модели выявить признаки, идентифицирующие отдельных учеников, и использовать эту информацию для получения точных предсказаний (в этом датасете ученик, как правило, получал одинаковые оценки за все свои сочинения).

Если бы мы попытались применить эту модель предсказания оценок, то она не смогла бы выдать точные оценки для новых учеников и просто выводила бы среднестатистические оценки тех, чьи сочинения использовались на этапе обучения. Это сложно назвать полезными результатами.

Чтобы устранить утечку данных в описанном случае, мы применили способ разделения данных: не на уровне сочинений, а на уровне учеников. Таким образом, данные каждого отдельного ученика теперь находились либо исключительно в обучающем наборе, либо исключительно в валидационном. Сложность задачи существенно возросла, а точность модели снизилась. Однако поскольку задача обучения теперь гораздо лучше отражала то, что ждет систему в эксплуатации, эта новая модель была намного более полезной с практической точки зрения.

Засорение выборки может происходить незаметно при решении самых распространенных задач. Давайте возьмем в качестве примера сайт для бронирования

аренды квартир. На этом сайте используется модель для прогнозирования кликов. Учитывая запрос пользователя, модель определяет, по какому варианту квартиры он может кликнуть, и решает, какие объявления показать.

Для обучения этой модели может использоваться датасет, содержащий данные о количестве предыдущих бронирований пользователя, о том, какие квартиры ему предлагались и по каким из представленных вариантов он кликал. Эти данные обычно размещаются в рабочей базе данных, позволяющей запрашивать информацию по каждому отдельному пользователю. Однако если бы разработчики этого сайта решили создать такой датасет, просто запрашивая информацию из базы данных, то это, скорее всего, привело бы к утечке. Вы уже догадались, почему?

Чтобы показать, что в данном случае может пойти не так, я схематично представил на рис. 5.4 процесс получения предсказаний для конкретного пользователя. Сверху на рисунке показано, какие признаки может использовать модель для прогнозирования кликов в реальных условиях. Здесь новому пользователю без ранее сделанных бронирований предлагается определенная квартира. Снизу на рисунке показано, как эти признаки будут выглядеть спустя несколько дней, если разработчики будут извлекать данные из базы данных.



Рис. 5.4. Утечка данных может происходить по таким неявным причинам, как неиспользование версионирования данных

Обратите внимание на различие в значении признака `previous_bookings` (предыдущие бронирования), которое отражает активность пользователя с момента первого показа объявлений. Использование снимка состояния базы данных ведет

к тому, что в обучающий набор «просачивается» информация о последующей активности пользователя. Теперь мы знаем, что в конечном итоге пользователь забронирует пять квартир! Такая утечка может привести к тому, что модель, обучаемая с помощью показанной ниже информации, будет выдавать правильное предсказание на неправильных обучающих данных. Модель будет показывать высокую точность на сгенерированном датасете за счет использования данных, к которым у нее не будет доступа при эксплуатации. В результате после развертывания модель будет работать хуже, чем ожидалось.

Из данного примера можно сделать следующий вывод: всегда критически исследуйте результаты модели, особенно если она демонстрирует удивительно высокую эффективность.

Разделение данных МО-редактора

Для обучения нашего МО-редактора мы используем датасет с вопросами и ответами с сайтов сети Stack Exchange. На первый взгляд может показаться, что здесь вполне уместно случайное разделение данных, которое легко реализовать с помощью библиотеки `scikit-learn`. Например, мы могли бы использовать вот такую функцию:

```
from sklearn.model_selection import train_test_split

def get_random_train_test_split(posts, test_size=0.3, random_state=40):
    """
    Извлекает обучающий и тестовый наборы из датафрейма
    Предполагается, что датафрейм содержит по одной строке для
    каждого образца вопроса
    :param posts: все посты с соответствующими метками
    :param test_size: пропорция, выделяемая для тестирования
    :param random_state: random seed
    """
    return train_test_split(
        posts, test_size=test_size, random_state=random_state
    )
```

Однако при таком подходе теоретически существует возможность утечки данных; вы уже догадались, как это может произойти?

Если мы вспомним, что представляет собой наш юзкейс, то увидим, что наша модель должна выдавать рекомендации для незнакомых ей вопросов исключительно на основе их содержания. Однако на сайте, посвященном вопросам и ответам, на успешность ответа влияет и множество других факторов. Одним из таких факторов является автор вопроса.

Если мы разделим данные случайным образом, то вопросы от определенного автора могут оказаться и в обучающем, и в валидационном наборе. Если вопро-

сы от популярных авторов будут обладать некоторым отличительным стилем, это может привести к тому, что модель подстроится под этот стиль. При этом она будет демонстрировать искусственно завышенную производительность на валидационном наборе, реальной причиной которой будет утечка данных. Во избежание этого следует проследить за тем, чтобы вопросы от каждого отдельного автора находились либо исключительно в обучающем, либо исключительно в валидационном наборе. Мы имеем здесь дело с той же разновидностью утечки данных, что и в рассмотренном ранее примере с прогнозированием оценок учеников.

Воспользовавшись классом `GroupShuffleSplit` библиотеки `scikit-learn` и передавая методу разделения этого класса признак, представляющий уникальный идентификатор автора вопроса, мы можем гарантировать, что вопросы от каждого автора будут находиться только в одном из наборов данных.

```
from sklearn.model_selection import GroupShuffleSplit

def get_split_by_author(
    posts, author_id_column="OwnerUserId", test_size=0.3, random_state=40
):
    """
    Проводит разделение на обучающий и тестовый наборы,
    гарантируя, что данные от каждого автора будут находиться
    только в одном из полученных наборов
    :param posts: все посты с соответствующими метками
    :param author_id_column: название столбца, содержащего author_id
    :param test_size: пропорция, выделяемая для тестирования
    :param random_state: random seed
    """
    splitter = GroupShuffleSplit(
        n_splits=1, test_size=test_size, random_state=random_state
    )
    splits = splitter.split(posts, groups=posts[author_id_column])
    return next(splits)
```

Вы можете ознакомиться со сравнительным анализом обоих методов разделения в ноутбуке из `GitHub`-репозитория этой книги, посвященном разделению данных.

После того как вы разделите датасет, можно будет обучить модель на тренировочном наборе. О том, из каких частей должен состоять пайплайн обучения, было рассказано в разделе «Начинайте с простого пайплайна» на с. 61. В `GitHub`-репозитории книги также есть ноутбук с примером сквозного пайплайна по обучению модели. Далее мы еще проанализируем результаты этого пайплайна.

Итак, мы уже разобрались с тем, какие риски следует учитывать при разделении данных, но что делать после обучения модели? В следующем разделе мы рассмотрим несколько практических подходов к оценке обученных моделей и наилучшие способы их применения.

Оценка производительности

Теперь, когда мы разделили свои данные, мы можем обучить модель и оценить ее производительность. В большинстве случаев при обучении модели ставится цель минимизировать значение функции потерь, отражающее, насколько отличаются предсказания модели от реальных меток. Чем меньше будет значение функции потерь, тем больше модель будет подстраиваться под данные. Как будет выглядеть минимизируемая функция, зависит от того, с какой именно моделью и задачей вы имеете дело, однако следует заметить, что в большинстве случаев полезно рассмотреть ее значение и на обучающем, и на валидационном наборе.

Обычно это помогает оценить *соотношение между отклонением и дисперсией*, которое показывает, насколько хорошо модели удалось извлечь из данных обобщаемую информацию, не запомнив при этом детали обучающего набора.



Хотя предполагается, что вы уже знакомы со стандартными метриками классификации, на всякий случай я все же напому, что они собой представляют. В случае задач классификации метрика «ассигасу» (доля верных результатов) подразумевает долю образцов, для которых модель выдает верные предсказания, то есть это общая доля верных предсказаний, как истинно положительных, так и истинно отрицательных. В случае сильного дисбаланса в данных высокая доля верных предсказаний может скрывать за собой плохую модель. Так, если доля положительных случаев (меток) составляет 99%, то модель, всегда предсказывающая положительный класс, будет иметь долю верных результатов, равную 99%, будучи при этом совершенно непригодной для использования. Метрики «precision» (точность), «recall» (полнота) и «f1» служат для решения этой проблемы. «Precision» (точность) — это доля верно предсказанных положительных значений среди всех положительных предсказаний. «Recall» (полнота) — это доля истинно положительных предсказаний среди образцов, имеющих положительную метку. Метрика f1 — это среднее гармоническое точности и полноты.

В ноутбуке из GitHub-репозитория этой книги, посвященном обучению простой модели, демонстрируется, как можно обучить первую версию «случайного леса» с использованием векторов TF-IDF и признаков, выбранных нами в разделе «Признаки МО-редактора» на с. 117.

Вот как при этом выглядят показатели «ассигасу» (доля верных результатов), «precision» (точность), «recall» (полнота) и «f1», полученные для обучающего и валидационного наборов:

```
Training accuracy = 0.585, precision = 0.582, recall = 0.585, f1 = 0.581  
Validation accuracy = 0.614, precision = 0.615, recall = 0.614, f1 = 0.612
```

Взглянув на эти метрики, можно сразу же заметить следующее:

- Поскольку мы имеем датасет с двумя сбалансированными классами, то случайный выбор класса для каждого образца обеспечил бы нам 50% верных предсказаний. Полученная нами доля верных ответов составляет 61%, что выше результата случайного отбора.
- На валидационном наборе мы получили более высокую долю верных предсказаний по сравнению с обучающим набором. Это говорит о том, что наша модель хорошо работает на незнакомых данных.

Теперь давайте копнем чуть глубже и оценим производительность модели еще точнее.

Дилемма смещения-дисперсии

Низкая эффективность на обучающем наборе — признак большого смещения, или так называемого *недообучения* (underfitting). Это означает, что модель не смогла извлечь полезную информацию: она не способна показать хорошие результаты даже на тех элементах данных, для которых она уже получила метку.

Высокая производительность на обучающем наборе при низкой на валидационном является признаком большого разброса, или так называемого *переобучения* (overfitting), при котором модели удастся найти способ преобразования входных обучающих данных в выходные, но этот способ не работает на незнакомых данных.

Недообучение и переобучение представляют собой два крайних случая дилеммы смещения-разброса, которые показывают, как меняется характер ошибок модели по мере возрастания ее сложности. Как видно на рис. 5.5, по мере возрастания сложности модели увеличивается разброс и уменьшается смещение, а модель переходит от недообучения к переобучению.

В нашем случае видим, что производительность на валидационном наборе выше, чем на обучающем, а значит, модель не переобучена. Таким образом, мы можем повысить производительность путем повышения сложности модели или признаков. Чтобы обеспечить оптимальное соотношение между смещением и разбросом, необходимо найти баланс между снижением смещения, которое ведет к повышению производительности на обучающем наборе, и снижением разброса, которое ведет к повышению производительности на валидационном наборе (часто попутно ухудшая производительность на обучающем наборе).

Метрики производительности позволяют составить общее представление о работе модели, однако они не объясняют, что именно ведет к получению хороших или плохих результатов. Чтобы улучшить свою модель, мы должны копнуть чуть глубже.

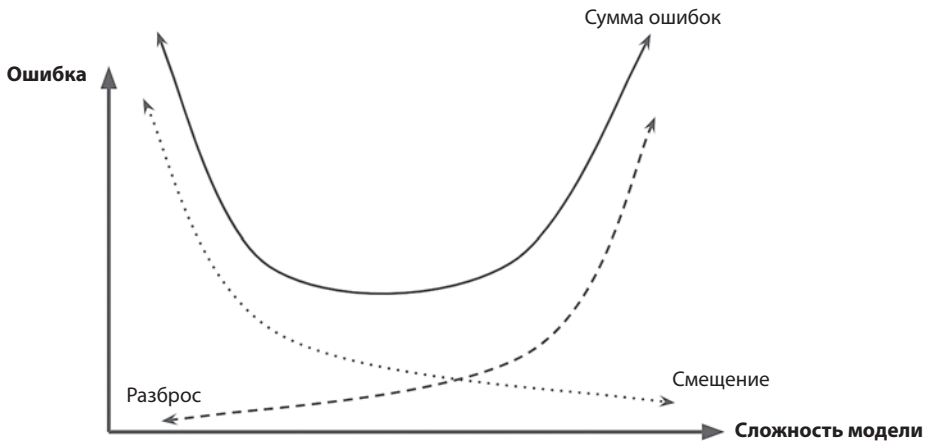


Рис. 5.5. По мере возрастания сложности модели увеличивается разброс и уменьшается смещение

Не ограничивайтесь агрегатными метриками

Метрики производительности позволяют установить, правильно ли обучилась модель на датасете или ее еще необходимо улучшить. Следующий шаг — проанализировать результаты и выяснить, почему модель работает хорошо или плохо. Это важно сделать, исходя из следующих двух соображений.

Валидация производительности

Метрики часто могут вводить в заблуждение. Так, при решении задачи классификации на данных со значительным дисбалансом, например для прогнозирования редкого заболевания, встречающегося у менее чем 1% пациентов, любая модель, которая будет всегда предсказывать, что пациент здоров, покажет точность (ассигасу), равную 99%, несмотря на отсутствие у нее какой-либо способности к прогнозированию. Хотя вы можете подобрать подходящие метрики производительности практически для любой задачи (так, для описанной выше задачи хорошо подойдет метрика $f1^1$), важно помнить о том, что это агрегатные метрики, не способные дать исчерпывающее представление о ситуации. Чтобы можно было доверять полученной оценке производительности, необходимо тщательно проанализировать результаты.

Итеративная доработка

Создание модели — итеративный процесс, и лучший способ начать его состоит в том, чтобы установить, что именно необходимо улучшить и как. Метрики

¹ <https://oreil.ly/fQAq9>

не позволяют вам выяснить, с чем плохо справляется модель и какие части пайплайна необходимо улучшить. Мне часто приходилось наблюдать, как специалисты по обработке данных пытались повысить производительность модели, просто пробуя применять разные модели или гиперпараметры либо бессистемно создавая дополнительные признаки. Такой подход равносителен бросанию дротиков в цель с завязанными глазами. Ключ к быстрому созданию эффективной модели — умение выявить и устранить конкретные причины плохой работы модели.

Исходя из этих соображений, давайте рассмотрим несколько способов более глубокого изучения производительности модели.

Оценка модели: не ограничивайтесь точностью

Существует бесконечное множество способов оценки производительности модели, и мы не будем рассматривать каждый из них. Обсудим здесь лишь те методы, которые лучше других помогают выяснить, что «скрывается под поверхностью».

При исследовании производительности вы как бы играете роль детектива, а методы, которые мы рассмотрим, можно считать разными способами поиска «улик». Мы начнем с изучения методов, позволяющих выявить интересные закономерности, сравнивая предсказания модели с данными.

Сравнение предсказаний с данными

Первый шаг к глубокой оценке модели — поиск способов, которые, в отличие от агрегатных метрик, позволяют более детально сравнить предсказания с данными. Вы должны разделить такие агрегатные метрики, как доля верных результатов (ассигасу), точность (precision) и полнота (recall), на несколько подмножеств данных. Давайте посмотрим, как это можно сделать для типичной в МО задачи — классификации.

Соответствующие примеры кода можно найти в GitHub-репозитории этой книги, в ноутбуке по сравнению данных.

Для задач классификации я обычно рекомендую начать с матрицы ошибок (рис. 5.6). Строки матрицы представляют реальные классы, а столбцы — предсказания модели. У модели, выдающей идеальные предсказания, эта матрица будет содержать нули во всех ячейках, за исключением диагонали, идущей из верхнего левого угла в нижний правый угол. На практике такое случается достаточно редко. Давайте посмотрим, почему матрица ошибок часто дает очень полезную информацию.

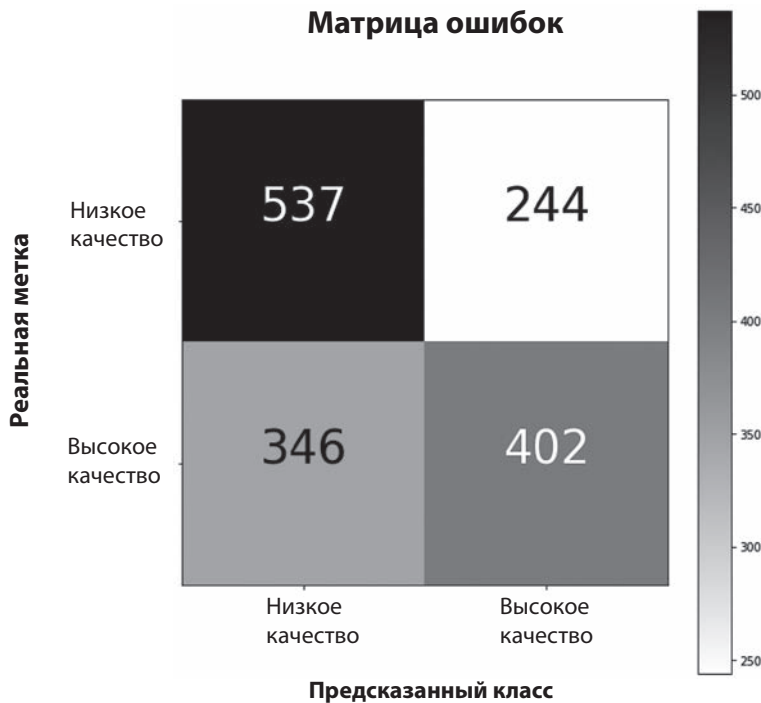


Рис. 5.6. Матрица ошибок исходной модели для классификации вопросов

Матрица ошибок

Матрица ошибок позволяет сразу увидеть, с какими классами модель работает эффективно, а какие классы вызывают у нее затруднения. Матрица особенно полезна в случае, когда датасет содержит большое количество классов или они не сбалансированы.

Мне много раз приходилось видеть, как у моделей с впечатляющей точностью один из столбцов в матрице ошибок оставался совершенно пустым. Это говорит о том, что существовал класс, который модель никогда не предсказывала. Такое часто можно наблюдать, если есть редкий класс, и иногда это не вызывает «тяжелых последствий». Однако если редкий класс представляет собой важный аспект, например случай невозврата заемщиком кредита, матрица ошибок позволит нам выявить эту проблему. Позже ее можно будет устранить, например путем присвоения редкому классу большего веса в функции потерь нашей модели.

На рис. 5.6 верхняя строка показывает, что обученная нами исходная модель хорошо выявляет низкокачественные вопросы, а нижняя строка — что модель плохо выявляет высококачественные. Действительно, модель правильно предсказывает класс лишь для половины вопросов с высоким баллом. Однако

взглянув на цифры в правом столбце, можно заключить, что когда модель классифицирует вопрос как высококачественный, этот результат, как правило, оказывается точным.

Матрица ошибок может оказаться еще более полезной, когда вы работаете более чем с двумя классами. Например, мне как-то довелось взаимодействовать с разработчиком, который пытался классифицировать произнесенные пользователем слова. Построив матрицу ошибок для своей последней модели, он сразу заметил две аномально высокие цифры, расположенные симметрично за пределами диагонали. Эти два класса (каждый из которых представлял слово) путали модель и были причиной большинства ее ошибок. Дальнейшее исследование показало, что этими сбивающими с толку словами были слова «*when*» (когда) и «*where*» (где). Чтобы заставить модель лучше различать эти сходные по звучанию слова, было достаточно собрать дополнительные данные для этих двух образцов.

Матрица ошибок позволяет сравнить предсказания модели с реальными метками каждого класса. Однако при отладке модели часто требуется рассмотреть не только предсказания, но и вероятности принадлежности к классу.

ROC-кривая

В случае задач бинарной классификации очень информативными являются кривые рабочей характеристики приемника (receiver operating characteristic, ROC). ROC-кривая показывает соотношение между долей истинно положительных результатов (true positive rate, TPR) и долей ложноположительных результатов (false positive rate, FPR).

подавляющее большинство классификационных моделей возвращает вероятности принадлежности образцов к тому или иному классу. Это означает, что во время инференса мы можем относить образец к тому или иному классу, если соответствующая вероятность выше определенного порогового уровня, который принято называть *порогом принятия решения*.

В большинстве классификаторов в качестве порога принятия решения по умолчанию используется вероятность 50%, однако это значение можно менять в зависимости от конкретного юзкейса. Систематически изменяя этот порог в диапазоне от 0 до 1 и измеряя TPR и FPR в каждой точке, мы получим ROC-кривую.

Располагая значениями вероятности предсказаний и реальными метками, мы можем легко вычислить TPR и FPR с помощью библиотеки `scikit-learn`, а затем построить ROC-кривую.

```
from sklearn.metrics import roc_curve

fpr, tpr, thresholds = roc_curve(true_y, predicted_proba_y)
```

Пример ROC-кривой показан на рис. 5.7. У таких кривых есть две особенности. Во-первых, диагональная линия, идущая из нижнего левого угла в верхний правый угол, представляет случайное угадывание класса. Это означает, что для того, чтобы превосходить минимальный уровень производительности (случайного угадывания), пара классификатор/порог должна находиться выше этой линии. Второй момент состоит в том, что случай идеальной работы модели здесь представляет зеленая пунктирная линия, проходящая через левый верхний угол.

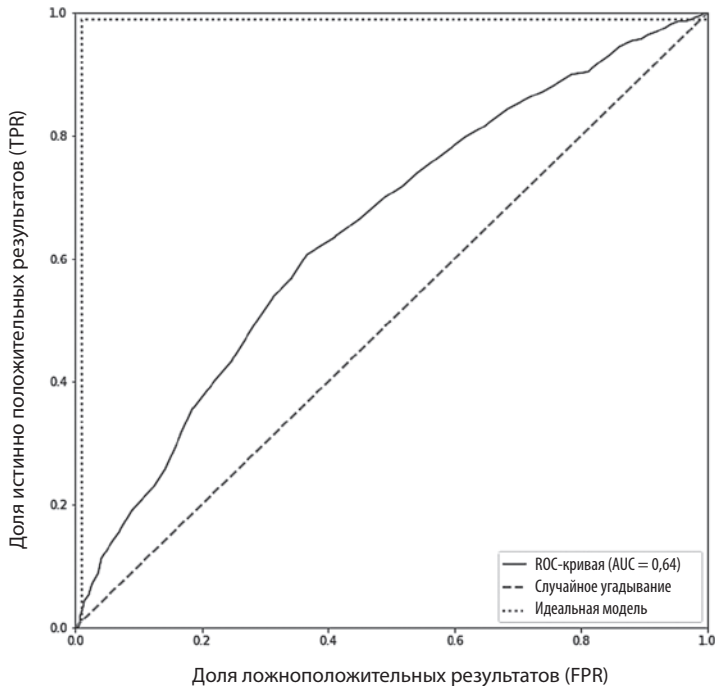


Рис. 5.7. ROC-кривая исходной модели

В силу этих двух особенностей для оценки производительности классификационных моделей часто используется такой показатель, как площадь под кривой (area under the curve, AUC). Чем выше показатель AUC, тем ближе классификатор к «идеальной» модели. У модели, производящей случайный отбор, показатель AUC будет равен 0,5, а у идеальной модели — 1. Однако для приложения, рассчитанного на практическое применение, мы должны выбрать конкретный порог, который даст наиболее целесообразное для нашего юзкейса соотношение между TPR и FPR.

Поэтому я рекомендую проводить на графике ROC-кривой вертикальные или горизонтальные линии, отражающие требования к продукту. Так, при создании системы, направляющей сотрудникам запросы пользователей в зависимости от

степени их срочности, допустимый уровень FPR будет полностью определяться численностью службы поддержки и количеством пользователей. Это означает, что даже не стоит рассматривать модели, у которых FPR превышает этот порог.

Отображение порога на графике ROC-кривой позволит вам не только получить максимальное значение AUC, но и более четко определить цель. Направьте ваши усилия на достижение этой цели!

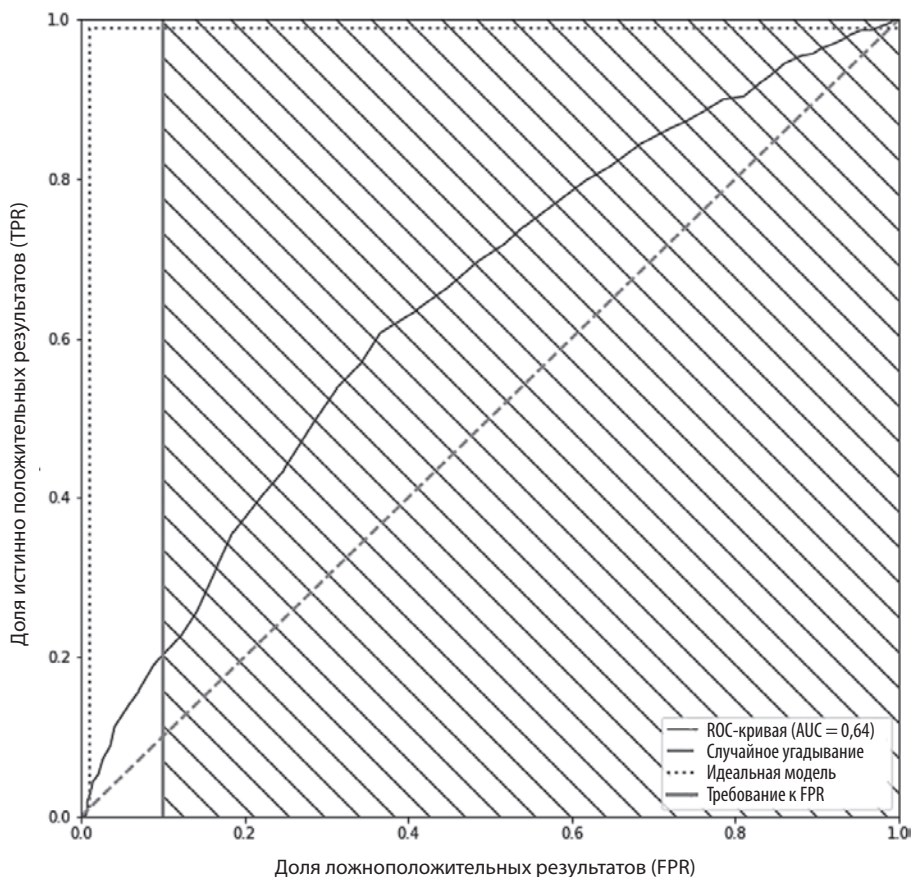


Рис. 5.8. Добавление на график ROC-кривой линий, отражающих требования к продукту

Модель нашего МО-редактора классифицирует вопросы как «хорошие» или «плохие». В этом контексте показатель TPR представляет долю качественных вопросов, правильно отнесенных нашей моделью к числу «хороших». Показатель FPR представляет долю «плохих» вопросов, отнесенных нашей моделью к числу «хороших». В том случае, когда мы не сможем помочь своим пользователям, необходимо убедиться, что мы по крайней мере не сделаем хуже. Это

означает, что мы не должны использовать модели, которые рискуют слишком часто рекомендовать плохо сформулированные вопросы. Таким образом, мы должны установить для FPR пороговый уровень, скажем, равный 10%, и использовать наилучшую из моделей, способных уложиться в это ограничение. На рис. 5.8 показано, как этот порог будет выглядеть на графике нашей ROC-кривой; он существенно сокращает площадь пространства, в котором может находиться допустимый порог принятия решения моделью.

ROC-кривая дает нам более конкретизированное представление о том, как изменится производительность модели, если мы сделаем ее предсказания более или менее консервативными. Еще один способ оценки вероятностей предсказаний модели — сравнить их распределения с распределениями реальных классов и посмотреть, насколько хорошо откалибрована модель.

Калибровочная кривая

Еще одним источником полезной информации для задач бинарной классификации являются калибровочные кривые, которые позволяют нам понять, насколько хорошо полученные вероятности отражают уверенность модели. Калибровочная кривая показывает соотношение между долей истинно положительных результатов и степенью уверенности классификатора.

Например, сколько из тех элементов данных, которые наш классификатор относит к положительным с уверенностью более 80%, действительно являются положительными? При идеальной модели калибровочная кривая представляет собой диагональную линию, идущую из нижнего левого угла в правый верхний.

Судя по верхнему графику рис. 5.9, наша модель хорошо откалибрована на участке со степенью вероятности от 0,2 до 0,7 и плохо — за пределами этого диапазона. Гистограмма прогнозируемых вероятностей (нижний график) показывает, что наша модель очень редко выдает вероятности, выходящие за пределы этого диапазона, что, по-видимому, и является причиной экстремальных значений на верхнем графике. Модель редко обладает высокой уверенностью в своих прогнозах.

При решении многих задач, таких как прогнозирование CTR рекламы, данные ведут к некоторому перекосу модели при приближении вероятности к 0 и 1, и калибровочная кривая позволяет сразу же это увидеть.

При анализе производительности модели также часто полезно визуализировать отдельные предсказания. Давайте посмотрим, как сделать эту визуализацию более эффективной.

Снижение размерности для анализа ошибок

Мы уже говорили о том, как можно использовать методы векторизации и снижения размерности для исследования данных, в разделе «Векторизация» на

с. 95 и в разделе «Снижение размерности» на с. 106. Теперь давайте посмотрим, как с помощью этих методов можно повысить эффективность анализа ошибок.

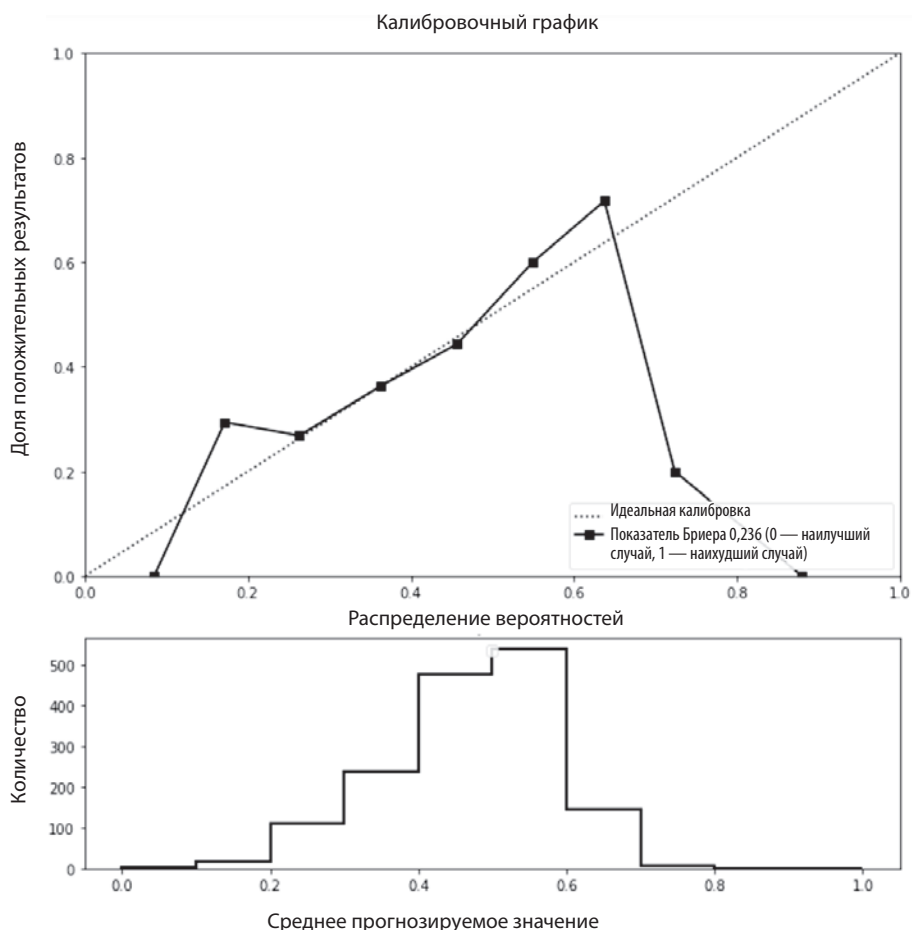


Рис. 5.9. Сверху: калибровочная кривая — диагональная линия представляет случай идеальной работы модели; снизу: гистограмма прогнозируемых значений

Когда мы в первый раз обсуждали использование методов снижения размерности для визуализации данных, мы окрашивали элементы данных в разные цвета в зависимости от класса, чтобы отобразить топологию меток. При анализе ошибок модели мы можем использовать цвета для выявления ошибок.

Чтобы выявить тренды появления ошибок, окрасьте элементы данных в разные цвета в зависимости от того, насколько правильным является предсказание модели. Это позволит вам выявить группы сходных элементов данных,

с которыми модель справляется плохо. Выявив такую область, отобразите несколько находящихся в ней элементов данных. Визуализация проблемных точек — отличный способ адаптировать модель под такие данные, выделив их признаки.

Еще одним вспомогательным средством при выявлении трендов в проблемных образцах могут служить методы кластеризации из раздела «Кластеризация» на с. 107. Кластеризовав данные, оцените производительность модели в каждом кластере и определите, в каких кластерах модель демонстрирует наихудшие результаты. Изучите элементы данных в этих кластерах; это поможет вам выделить дополнительные признаки.

Методы снижения размерности являются лишь одним из возможных способов выявления проблемных образцов. Еще один способ — напрямую использовать показатель уверенности модели.

Метод первых k элементов

Выявление областей с большим количеством ошибок помогает определить места сбоя модели. Выше было показано, как с помощью снижения размерности найти такие области, однако сделать это можно и напрямую. Используя вероятности предсказаний, мы можем установить, для каких элементов данных модели сложнее всего выдать результат или где уверенность в предсказании меньше. Давайте назовем этот подход *методом первых k элементов*.

Суть метода проста. Сначала нужно выбрать количество визуализируемых образцов. Обозначим это число как k . Если визуализацию для проекта выполняет один разработчик, лучше начать с десяти-пятнадцати образцов. Для каждого класса или кластера отобразите:

- k образцов с наиболее высокой производительностью;
- k образцов с наиболее низкой производительностью;
- k образцов с наименьшей степенью уверенности.

Давайте подробнее рассмотрим каждую из этих категорий.

k образцов с наиболее высокой производительностью

Прежде всего, отобразите k образцов, для которых модель выдала правильный результат и сделала это с наибольшей степенью уверенности. Посмотрите, нет ли у этих образцов сходства в значениях признаков. Это позволит вам установить, какие признаки ваша модель использует успешно.

Затем, чтобы установить, какие признаки модели не удастся использовать, отобразите образцы с низкой производительностью.

k образцов с наиболее низкой производительностью

Отобразите k образцов, для которых модель с высокой уверенностью выдала неправильный результат. Сначала сделайте это в обучающем, а затем — в валидационном наборе.

Как и визуализация кластеров ошибок, визуализация k образцов обучающего набора с низкой производительностью помогает установить, какие тренды есть в элементах данных, на которых модель дает сбой. Отобразив их, вы сможете определить дополнительные признаки, которые упростят работу модели.

Так, например, когда я изучил ошибки исходной модели МО-редактора, я обнаружил, что некоторые из вопросов имели низкий балл потому, что в действительности не были вопросами. Модель изначально не умела присваивать им низкий балл, поэтому я ввел в качестве дополнительного признака количество вопросительных знаков в тексте. Этот признак позволил модели выдавать точные предсказания для «ненастоящих» вопросов.

Визуализация k образцов с наиболее низкой производительностью из валидационного набора помогает выявить образцы, сильно отличающиеся от обучающих данных. Если вы обнаружите такие образцы в валидационном наборе, скорректируйте свою стратегию разделения данных, следуя рекомендациям, изложенным в разделе «Разделение датасета» на с. 128. Наконец, модели не всегда уверенно выдают правильный или неправильный результат; уверенность предсказания может быть и низкой. Давайте рассмотрим и этот случай.

k образцов с наименьшей уверенностью

Визуализация k образцов с наименьшей уверенностью сводится к отображению экземпляров данных, в предсказаниях которых модель сомневалась больше всего. Для классификационных моделей, которым уделяется основное внимание в этой книге, это такие образцы, которые модель практически с одинаковой уверенностью относит к разным классам.

Если модель хорошо откалибрована (см. описание процесса калибровки в разделе «Калибровочная кривая» на с. 145), то она будет выдавать одинаковые вероятности для тех образцов, которые даже человек классифицирует с трудом. Так, модель для классификации кошек и собак отнесет к этой категории изображения, на которых одновременно присутствуют и кошка, и собака.

Наличие таких образцов в обучающем наборе часто является признаком конфликта меток. Если обучающий набор будет содержать два абсолютно одинаковых или очень похожих образца с метками разных классов, то модель минимизирует потери на этапе обучения, выдавая одинаковую вероятность принадлежности к разным классам. Таким образом, конфликт меток ведет к низкой уверенности в предсказаниях, и вы можете попытаться выявить такие образцы, используя метод первых k элементов.

Отображение k образцов с наименьшей уверенностью для валидационного набора часто позволяет выявить пробелы в обучающих данных. Низкая уверенность на валидационных образцах (классификация которых не представляет проблем для человека) часто объясняется тем, что модель не сталкивалась с такими данными в обучающем наборе. В силу этого отображение k образцов валидационной выборки часто позволяет установить, какие данные следует добавить в обучающий набор.

Оценить первые k элементов просто. Поделюсь реализацией рабочего метода.

Советы по реализации метода первых k элементов

Ниже представлен пример простейшей реализации метода первых k элементов с использованием структуры DataFrame библиотеки pandas. Эта функция принимает в качестве входных данных датафрейм, содержащий прогнозируемые вероятности и метки, и возвращает по k первых элементов для каждой категории. Код можно найти в GitHub-репозитории книги.

```
def get_top_k(df, proba_col, true_label_col, k=5, decision_threshold=0.5):
    """
    Для задач бинарной классификации
    Возвращает по k образцов с наиболее высокой и наиболее низкой
    производительностью для каждого класса
    Также возвращает k образцов с наименьшей уверенностью
    :param df: датафрейм, содержащий предсказания и реальные метки
    :param proba_col: название столбца с вероятностями
    :param true_label_col: название столбца с реальными метками
    :param k: количество отображаемых образцов для каждой категории
    :param decision_threshold: порог принятия решения классификатора
    для положительной классификации образца
    :return: correct_pos, correct_neg, incorrect_pos,
    incorrect_neg, most_uncertain
    """
    # Получаем правильные и неправильные предсказания
    correct = df[
        (df[proba_col] > decision_threshold) == df[true_label_col]
    ].copy()
    incorrect = df[
        (df[proba_col] > decision_threshold) != df[true_label_col]
    ].copy()

    top_correct_positive = correct[correct[true_label_col]].nlargest(
        k, proba_col
    )
    top_correct_negative = correct[~correct[true_label_col]].nsmallest(
        k, proba_col
    )
    top_incorrect_positive = incorrect[incorrect[true_label_col]].nsmallest(
        k, proba_col
    )
    top_incorrect_negative = incorrect[~incorrect[true_label_col]].nlargest(
```

```

    k, proba_col
)

# Получаем образцы, наиболее близкие к порогу принятия решения
most_uncertain = df.iloc[
    (df[proba_col] - decision_threshold).abs().argsort()[:k]
]

return (
    top_correct_positive,
    top_correct_negative,
    top_incorrect_positive,
    top_incorrect_negative,
    most_uncertain,
)
```

Давайте наглядно рассмотрим применение метода первых k элементов на примере нашего МО-редактора.

Использование метода первых k элементов для МО-редактора

Давайте применим этот метод к тому первому классификатору, который мы обучили. Примеры использования метода можно найти в соответствующем ноутбуке в GitHub-репозитории этой книги.

На рис. 5.10 показано по два образца каждого класса с наиболее точными предсказаниями первой модели МО-редактора. Сильнее всего у этих двух классов различается признак `text_len` (длина текста). Наш классификатор «выучил», что хорошо сформулированные вопросы имеют тенденцию быть длинными, а плохо сформулированные — короткими. При отнесении образцов к первому или второму классу модель в значительной мере полагается на длину текста.



Рис. 5.10. k образцов с наиболее точными результатами

Рисунок 5.11 подтверждает это предположение. Неотвеченные вопросы, которые наш классификатор уверенно относит к числу ответченных, обладают наибольшей длиной, и наоборот. Это наблюдение также согласуется с анализом, проведенным в разделе «Оценка важности признаков» на с. 152, который показывает, что `text_len` является наиболее важным признаком.

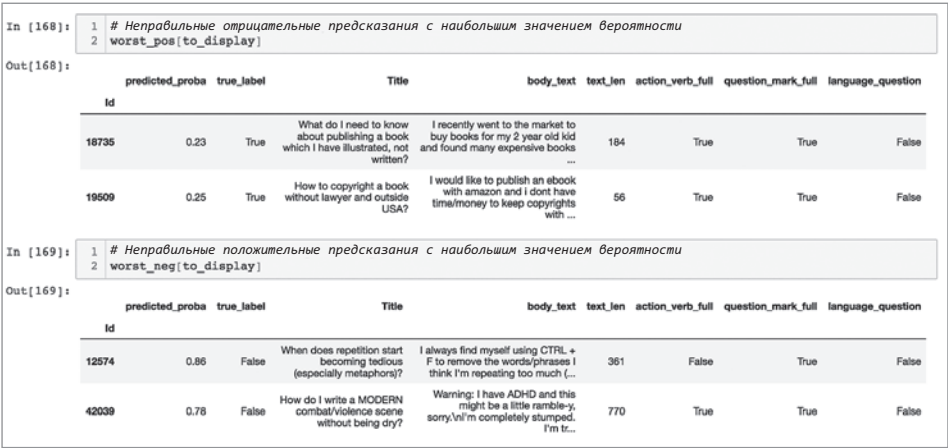


Рис. 5.11. k образцов с наименее точными результатами

Таким образом, мы выяснили, что классификатор использует признак `text_len`, чтобы легко распознать ответченные и неотвеченные вопросы, однако одного признака недостаточно, это может привести к неправильной классификации. Для улучшения модели нам нужно выделить дополнительные признаки. Если мы отобразим на графике более двух образцов, то поймем, какие еще признаки мы могли бы использовать.

Применение метода первых k элементов и на обучающих, и на валидационных данных позволяет выявить ограничения модели и набора данных. Мы рассмотрели, как с помощью этого метода можно определить, достаточно ли сбалансирован датасет, хватает ли в нем репрезентативных образцов и способна ли модель верно отразить имеющиеся данные.

Мы говорили главным образом о методах оценки классификационных моделей, поскольку такие модели могут применяться для решения широкого круга задач. Давайте вкратце поговорим о том, как оценить производительность, когда вы имеете дело не с классификацией.

Другие модели

Многие модели можно оценивать по той же схеме, что и классификацию. Так, например, если модель должна отображать ограничительную рамку вокруг обна-

руженных на изображении объектов, обычно используется такой показатель, как точность (ассигасу). Поскольку на каждом изображении может быть несколько ограничительных рамок, представляющих объекты и предсказания, вычисление точности будет включать в себя один дополнительный шаг. Сначала нужно будет рассчитать, насколько предсказания модели схожи с метками (это часто делается с использованием коэффициента Жаккара¹), и на основе этого пометить каждое предсказание как правильное или неправильное. Потом можно рассчитать точность и использовать все те методы, которые мы рассмотрели ранее в этой главе.

Сходным образом при создании моделей для рекомендаций контента наилучший подход к итеративной доработке — протестировать модель на различных категориях и оценить производительность на каждой из них. При этом модель оценивается так же, как и в случае классификации, с тем лишь отличием, что в качестве классов здесь выступают категории.

Если такие методы применить сложно, например когда мы имеем дело с порождающими моделями, вы все равно можете исследовать датасет и разделить его на несколько категорий, а затем сгенерировать метрики производительности для каждой категории.

Мне как-то довелось работать со специалистом по обработке данных, решившим создать модель для упрощения предложений. Когда мы посмотрели, как производительность модели зависит от длины предложений, то увидели, что модели гораздо труднее работать с длинными предложениями. Нам потребовалось потратить много усилий на исследование и ручную разметку, но зато мы поняли, что на следующем шаге мы должны дополнить обучающие данные более длинными предложениями. В итоге это привело к существенному повышению производительности.

Мы рассмотрели различные способы оценки производительности модели путем сопоставления ее предсказаний с метками, однако это можно сделать и с помощью непосредственного анализа модели. Когда модель вообще не показывает хороших результатов, часто бывает полезно попытаться интерпретировать ее предсказания.

Оценка важности признаков

Еще один способ анализа производительности модели состоит в том, чтобы посмотреть, на основе каких признаков данных она делает свои предсказания. Оценка важности признаков позволяет исключить или доработать те признаки, которые в данный момент не приносят модели никакой пользы. Такая оценка также позволяет выявить признаки, имеющие подозрительно высокую про-

¹ <https://oreil.ly/eklQm>

гностическую значимость, что часто является свидетельством утечки данных. Сначала мы рассмотрим, как можно оценить важность признаков для моделей, легко поддающихся такому анализу, а затем коснемся случаев, когда оценку важности признаков трудно провести напрямую.

Непосредственная оценка классификатора

Чтобы убедиться в том, что модель работает правильно, отобразите графически, какие признаки модель принимает в расчет, а какие — нет. Если модель простая, например регрессия или решающие деревья, для оценки важности признаков достаточно просто посмотреть, какие параметры модель «выучила».

Первая модель для МО-редактора — случайный лес. Здесь можно просто отобразить список всех признаков, ранжированный по их важности. Для этого будем использовать API библиотеки `scikit-learn`. Код для оценки важности признаков, вместе с примерами его использования, можно найти в соответствующем ноутбук в GitHub-репозитории этой книги.

```
def get_feature_importance(clf, feature_names):
    importances = clf.feature_importances_
    indices_sorted_by_importance = np.argsort(importances)[::-1]
    return list(
        zip(
            feature_names[indices_sorted_by_importance],
            importances[indices_sorted_by_importance],
        )
    )
```

После применения функции к обученной модели и несложной обработки списка мы получаем десять наиболее информативных признаков:

```
Top 10 importances:
text_len: 0.0091
are: 0.006
what: 0.0051
writing: 0.0048
can: 0.0043
ve: 0.0041
on: 0.0039
not: 0.0039
story: 0.0039
as: 0.0038
```

Здесь следует обратить внимание на следующие моменты:

- Наиболее информативным признаком является длина текста.
- Важность других созданных нами признаков на порядок меньше. Модели не удалось их использовать для осмысленного выделения классов.

- Другие признаки представляют собой либо очень распространенные слова, либо существительные, имеющие отношение к теме текста.

Поскольку наша модель и признаки просты, эти результаты могут дать нам идеи о том, какие еще признаки мы могли бы создать. Мы могли бы, например, добавить признак, отражающий частоту использования распространенных и редких слов, и посмотреть, предсказывают ли эти признаки, что вопрос получит высокий балл.

Если признаки или модели становятся сложными, то для оценки важности признаков необходимо использовать инструменты объяснения модели.

Интерпретаторы «черного ящика»

По мере усложнения признаков их важность все труднее определить. Если используется сложная модель, например нейронная сеть, часто это совсем невозможно сделать. В таких случаях полезны инструменты, объясняющие предсказания модели безотносительно к ее внутреннему устройству.

Обычно такие интерпретаторы выявляют прогностически значимые признаки для определенного элемента данных, а не для всего датасета. Это делается путем изменения значений каждого признака определенного образца и наблюдения за тем, как изменятся предсказания модели. Сегодня широкой известностью пользуются методы LIME¹ и SHAP².

Исчерпывающий пример использования этих методов можно найти в соответствующем ноутбуке в GitHub-репозитории книги.

На рис. 5.12 показан пример объяснения с помощью метода LIME того, какие слова больше всего повлияли на отнесение конкретного образца к числу высококачественных вопросов. LIME сгенерировал это объяснение путем поочередного удаления слов из исходного вопроса.

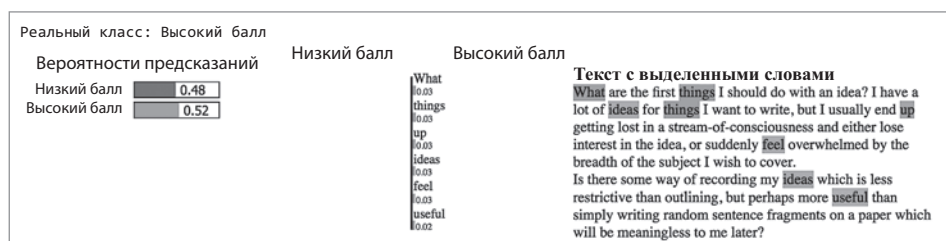


Рис. 5.12. Объяснение на основе конкретного образца

¹ <https://github.com/marcotcr/lime>

² <https://github.com/slundberg/shap>

Как мы видим, модель правильно предсказала, что данный вопрос получит высокий балл, но степень уверенности в этом составила лишь 52%. Справа на рис. 5.12 показано, какие слова оказали наибольшее влияние на предсказание. Похоже, эти слова не имеют особого отношения к качеству вопроса, поэтому давайте возьмем еще несколько примеров и посмотрим, использует ли модель более полезные закономерности.

Чтобы получить общее представление о трендах, мы можем применить LIME на более крупной выборке вопросов. Запуская LIME для каждого вопроса и накапливая совокупные результаты, мы можем установить, какие слова модель считает в целом прогностически значимыми при принятии решений.

На рис. 5.13 показано, какие слова были наиболее важными при генерировании предсказаний для 500 вопросов из нашего датасета. Как мы видим, на этой более крупной выборке также очевидна тенденция модели использовать распространенные слова. Похоже, что никакие другие закономерности нашей модели выделить не удастся. При использовании «мешка слов» (bag of words) признаки, представляющие редкие слова, часто имеют практически нулевое значение.

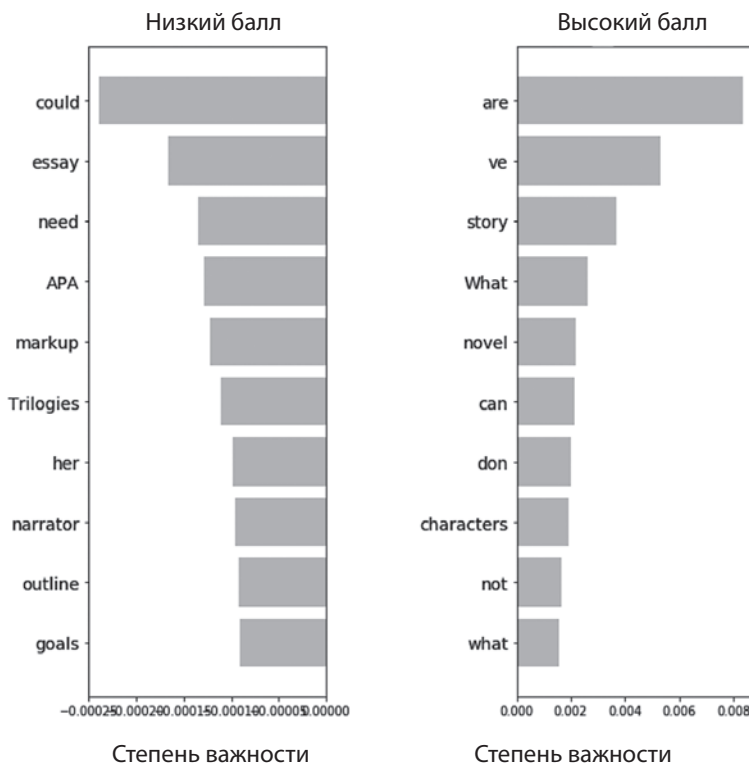


Рис. 5.13. Объяснение на основе нескольких образцов

Чтобы исправить это, мы можем либо собрать более крупный датасет и расширить словарь, предоставляемый модели, либо создать признаки, которые будут еще полезнее.

Вероятно, вас удивит, какие предикторы использует модель. Если какие-либо признаки обладают подозрительно высокой прогностической значимостью, попробуйте найти образцы с этими признаками в обучающих данных и изучите их. Используйте это как повод лишний раз проверить, насколько правильно вы разделили датасет и нет ли у вас утечки данных.

Например, мне как-то довелось курировать работу инженера по МО, который решил создать модель для автоматической классификации электронных писем по различным темам на основе их содержания. Он обнаружил, что наилучшим предиктором для его модели был трехбуквенный код, который присутствовал в начале каждого письма. Как оказалось, это был внутренний код датасета, который практически полностью соответствовал меткам. Модель полностью игнорировала содержание электронного письма и просто запоминала уже имеющиеся метки. Это очень наглядный пример утечки данных, выявленной лишь на этапе оценки важности признаков.

Заключение

Эту главу мы начали с рассмотрения критериев выбора исходной модели на основе полученной к этому моменту информации. Затем поговорили о важности разделения датасета на несколько выборок и узнали, как исключить утечку данных.

После обучения исходной модели мы разобрались, как оценить ее производительность, используя различные способы сопоставления предсказаний с данными. Наконец, мы узнали, как оценить саму модель с помощью определения важности каждого признака и использования интерпретаторов «черного ящика» для понимания, на основе каких признаков модель делает свои предсказания.

К этому моменту у вас уже должны появиться какие-то идеи о том, как улучшить вашу модель. Это подводит нас к теме главы 6, где мы подробно поговорим о том, как решить выявленные здесь проблемы путем поиска и устранения неисправностей в пайплайне МО.

Отладка МО-приложения

В предыдущей главе мы обучили свою первую модель и оценили ее производительность.

Доведение пайплайна до приемлемого уровня производительности — непростая задача, требующая последовательных итераций. Данная глава проведет вас через один такой итерационный цикл. Я расскажу об инструментах отладки пайплайна моделирования и способах написания тестов для проверки работоспособности пайплайна после того, как мы вносим в него изменения.

При разработке программного обеспечения рекомендуется регулярно тестировать и валидировать код, особенно на таких деликатных этапах, как обеспечение безопасности или парсинг входных данных. В особенности это касается приложений на базе МО, где ошибки выявить гораздо труднее, чем в традиционном ПО.

Мы рассмотрим здесь несколько советов, которые помогут убедиться, что ваш пайплайн надежен и его можно опробовать, не вызвав сбой всей системы. Но сначала давайте чуть подробнее остановимся на лучшей практике программной разработки.

Передовые практики разработки ПО

В работе над большинством проектов МО вы будете раз за разом создавать модель, анализировать ее недостатки и затем устранять их. Во многих случаях вам также придется неоднократно изменять компоненты своей инфраструктуры. Поэтому очень важно разобраться с тем, как повысить скорость выполнения итераций.

Как и в случае разработки любого другого ПО, в сфере МО необходимо придерживаться проверенных временем практик программной разработки. Большинство этих рекомендаций может применяться к проектам МО в их

обычном виде, как, например, принцип KISS («keep it stupid simple», «чем проще, тем лучше»)¹.

Проекты МО носят итеративный характер и требуют многократного повторения операций очистки данных, создания признаков и выбора модели. Даже при использовании передовых практик разработки скорость выполнения итераций часто снижается из-за отладки и тестирования. Ускорение процесса отладки и написания тестов может существенно ускорить разработку любого ПО. Это еще более актуально для проектов МО, где в силу стохастического характера моделей поиск и устранение простейшей ошибки порой занимают целый день.

Существует много обучающих ресурсов по теме отладки обычных программ, в частности краткое руководство по отладке от Чикагского университета². Если вы, как и большинство других специалистов по МО, предпочитаете писать код на языке Python, то я рекомендую вам ознакомиться с документацией по интерактивному отладчику pdb³ из стандартной библиотеки языка Python.

В то же время, в отличие от большинства обычных программ, МО-код часто может выдавать совершенно абсурдные результаты при, казалось бы, корректной работе. Это означает, что, хотя все эти инструменты и рекомендации применимы к большинству МО-программ, их недостаточно для диагностики распространенных ошибок. Это иллюстрирует рис. 6.1: в то время как в большинстве обычных приложений максимальное покрытие кода тестами дает высокую уверенность в надлежащем функционировании приложения, МО-пайплайн может выдавать совершенно неверные результаты даже при успешном выполнении большого количества тестов. МО-программа должна не только работать без ошибок, но и делать точные прогнозы.

Обычное ПО	ПО на базе МО
Отсутствие сбоев	Отсутствие сбоев
Успешное выполнение тестов	Успешное выполнение тестов
✓ Модульные тесты	✓ Модульные тесты
✓ Интеграционные тесты	✓ Интеграционные тесты
✓ Регрессионные тесты	✓ Регрессионные тесты
80% уверенности в качестве приложения	✓ Тесты распределения
	✓ Тесты модели на исторических данных
	Доля верных результатов, точность, полнота
	20% уверенности в успешности приложения

Рис. 6.1. Даже при отсутствии ошибок пайплайн МО не всегда дает необходимый результат

¹ <https://oreil.ly/ddzav>

² <https://oreil.ly/xwfYn>

³ <https://oreil.ly/CBldR>

Учитывая то, что при отладке МО-кода может возникать ряд дополнительных проблем, давайте рассмотрим конкретные методы, которые могут помочь в их решении.

Передовые практики для МО-приложений

Когда речь идет о МО-приложениях, то здесь (в отличие от других типов ПО) недостаточно того, что программа выполняется, нужно убедиться в правильности выдаваемых результатов. Даже при полном отсутствии ошибок ваш пайплайн может создавать совершенно бесполезную модель.

Допустим, ваша программа загружает данные и передает их модели. Модель принимает эти входные данные и оптимизирует свои параметры в соответствии с алгоритмом обучения. И наконец, обученная вами модель выдает результаты на основе некоторого другого набора данных. Вы убедились, что программа работает без каких-либо видимых ошибок. Однако проблема в том, что это ни в коей мере не гарантирует, что ваша модель выдает правильные предсказания.

Большинство моделей просто принимает числовые входные данные в определенном формате (например, матричное представление изображения) и выдает в качестве результата данные в другом формате (например, список координат ключевых точек входного изображения). Это означает, что большинство моделей будет работать, даже если операция обработки данных будет искажать данные перед передачей их модели, при условии, что это будут числовые данные в воспринимаемом моделью формате.

А если пайплайн моделирования выдает плохие результаты, как можно определить, что послужило тому причиной — низкое качество модели или наличие ошибки на предшествующих этапах обработки?

Лучший способ решения этих проблем — работать поэтапно. Сначала проверьте поток данных, затем — способность модели обучаться и, наконец, операции обобщения и инференса. Общая схема этого процесса представлена на рис. 6.2.

Мы последовательно пройдемся по каждому из трех этапов и подробно рассмотрим все их составляющие. При появлении сложных ошибок часто возникает искушение пропустить некоторые шаги, однако, как показывает мой опыт, строгое следование этому подходу — самый быстрый способ выявить и исправить ошибки.

Давайте начнем с проверки потоков данных. Самый простой способ — взять очень небольшое подмножество данных и убедиться в том, что эти данные могут пройти через весь ваш пайплайн.

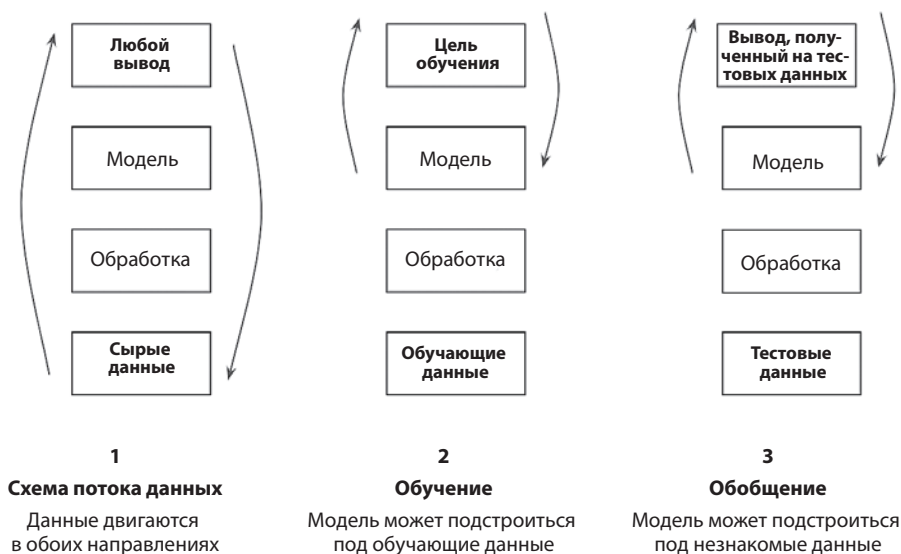


Рис. 6.2. Последовательность отладки пайплайна

Отладка потока данных: визуализация и тестирование

Первый этап несложен и может существенно упростить вам жизнь после того, как вы примете его на вооружение: сначала нужно убедиться в том, что пайплайн работает для небольшого подмножества образцов из вашего датасета. На рис. 6.2 этот этап обозначен как схема потока данных. После того как вы убедитесь, что пайплайн работает для нескольких образцов, можно будет написать тесты для проверки работоспособности пайплайна по мере внесения в него изменений.

Начните с одного образца

Цель первого этапа — убедиться, что вы можете получить данные, преобразовать их в нужный формат, передать модели и получить на выходе некоторый корректный результат. Пока оставьте без внимания вопрос о том, может ли модель чему-либо научиться; нужно лишь проверить, могут ли проходить данные через ваш пайплайн.

Конкретные шаги:

- отберите несколько образцов из своего датасета;
- убедитесь, что модель выдает предсказания для этих образцов;

- убедитесь, что модель корректирует свои параметры для получения правильных предсказаний для этих образцов.

Первые две операции служат для проверки того, что модель способна принять входные данные и выдать приемлемый результат. Этот первоначальный результат, скорее всего, будет неправильным с точки зрения моделирования, но вы убедитесь, что данные проходят через весь пайплайн.

Третья операция позволяет проверить, способна ли модель научиться сопоставлять входные и выходные данные. Конечно, если элементов данных немного, то, как правило, модель на них переобучается. С помощью этой операции вы должны просто убедиться в том, что модель может скорректировать свои параметры для получения более точных результатов.

Пример: если вы хотите научить свою модель предсказывать, будет ли успешным проведение кампании на краудфандинговой платформе Kickstarter, то, вероятно, обучать модель вы будете на данных по всем кампаниям, проведенным за последние несколько лет. Следуя рекомендации выше, сначала убедитесь, может ли ваша модель выдать предсказания для двух кампаний. Затем, используя метки этих двух кампаний (данные о том, были ли они успешными), оптимизируйте параметры модели до тех пор, пока она не начнет выдавать правильные результаты.

Если вы правильно выбрали модель, то она будет способна обучаться на имеющемся наборе данных. И если модель способна обучаться на всем датасете, то она сможет обучиться и на отдельном элементе данных. Таким образом, способность обучаться на нескольких образцах является необходимым условием для того, чтобы модель могла обучаться на всем датасете. Кроме того, это гораздо проще, чем обучать на целом наборе данных. Начав с одного образца, мы можем быстро сузить круг потенциальных проблем.

подавляющее большинство ошибок, возникающих на этом начальном этапе, связано с несоответствием данных: загруженные и предобработанные данные поступают в модель в неприемлемом для нее формате. Так, например, поскольку большинство моделей принимает только числовые значения, они могут дать сбой, если датасет содержит пропуски.

Иногда несоответствие данных остается незамеченным и в итоге ведет к неявным ошибкам. Часто при скормливании пайплайну значений, выходящих за рамки допустимого диапазона или неподходящего формата, он продолжает работать, но выдает в итоге плохую модель. Модели, которым требуются нормированные данные, часто могут обучаться и на ненормированных — у них просто не получается извлечь из этого какую-либо пользу. Точно так же при «скармливании» матрицы неправильного формата модель может просто неверно интерпретировать входные данные и выдать неправильные результаты.

Такие ошибки труднее поддаются выявлению, поскольку они обычно проявляются на более поздних стадиях процесса, когда мы оцениваем производитель-

ность модели. Лучший способ заранее обнаружить подобные ошибки — визуализировать данные по мере создания пайплайна и написать тесты для проверки ваших предположений. Давайте посмотрим, как это сделать.

Визуализация

Как было показано в предыдущих главах, не только метрики, но также регулярная проверка и исследование данных являются важной составляющей моделирования. Рассмотрев для начала лишь несколько образцов, вы сможете легко заметить имеющиеся изменения или несоответствия.

Цель такого процесса — регулярный контроль происходящих изменений. Если рассматривать пайплайн данных как своего рода «сборочный конвейер», то вы должны проверять состояние своего «продукта» *после каждого серьезного изменения*. Это не значит, что нужно проверять значение данных буквально на каждом шагу, но проверка только на входе и выходе будет явно неинформативной.

На рис. 6.3 показано несколько стандартных контрольных точек, на которых вы могли бы проверять пайплайн данных. В этом примере мы проверяем состояние данных на нескольких этапах, начиная с загрузки сырых данных и заканчивая выдаваемым в итоге результатом.

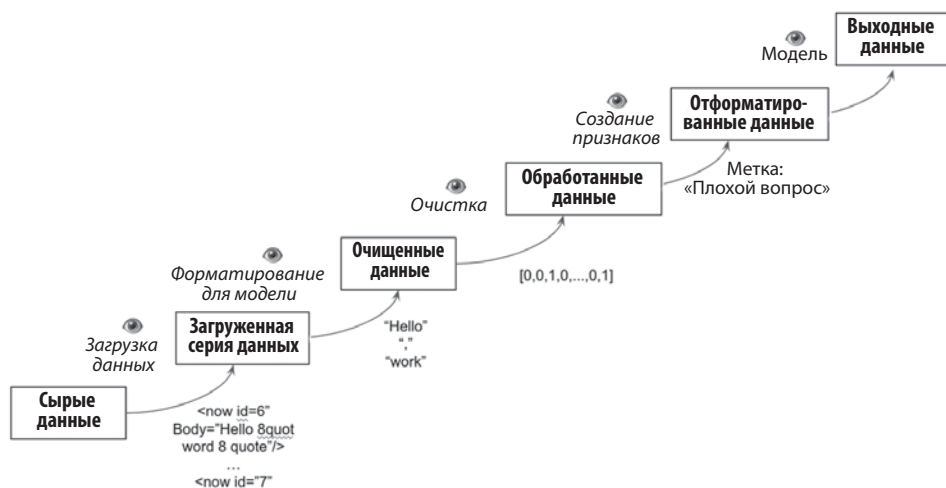


Рис. 6.3. Контрольные точки

Теперь давайте подробнее обсудим ключевые этапы, которые часто нуждаются в проверке. Мы последовательно рассмотрим загрузку и очистку данных, создание признаков, форматирование и вывод.

Загрузка данных

Вне зависимости от того, как вы загружаете данные — с диска или через API, вам нужно проследить за тем, чтобы они были в нужном формате. Это делается почти так же, как при проведении разведочного анализа данных, только здесь данные проверяются в контексте созданного вами пайплайна, чтобы не допустить искажения вследствие каких-либо ошибок.

Содержат ли данные все необходимые поля? Содержат ли какие-то поля нулевые или неизменяющиеся значения? Не выходят ли какие-то значения за пределы допустимого диапазона, как, например, в случае, когда признак возраста содержит отрицательное значение? Если вы работаете с текстом, речью или изображениями, соответствуют ли образцы вашим представлениям о том, как они должны выглядеть или звучать?

Большинство этапов обработки опирается на некоторые предположения о структуре входных данных, поэтому принципиально важно, чтобы вы проверили этот аспект.

Чтобы выявить несоответствия между ожидаемыми и реальными данными, иногда требуется отобразить больше, чем один или два элемента. Визуализация репрезентативной выборки позволит исключить вероятность того, что вам попадется некоторый «удачный» образец и вы ошибочно посчитаете, что таким же качеством обладают и все остальные элементы данных.

На рис. 6.4 показано, как может быть выполнена визуализация для нашего учебного примера — см. ноутбук по теме исследования данных в GitHub-репозитории книги. Сотни экземпляров в нашем архиве не имеют данных о типе поста и поэтому должны быть удалены из обучающего датасета: на рисунке видны строки со значением 5 в столбце PostTypeId, а этот признак не описан в документации датасета.

In [5]: df[df["Body"].isna()]

wmedDate	CreationDate	FavoriteCount	LastActivityDate	LastEditDate	...	ParentId	PostTypeId	Score	Tags	Title	ViewCount	body_text	text_len	tokenized	is_question
NaN	2011-03-22T19:49:56.600	NaN	2011-03-22T19:49:56.600	2011-03-22T19:49:56.600	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-22T19:51:05.897	NaN	2011-03-22T19:51:05.897	2011-03-22T19:51:05.897	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T19:35:10.353	NaN	2011-03-24T19:35:10.353	2011-03-24T19:35:10.353	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T19:41:38.677	NaN	2011-03-24T19:41:38.677	2011-03-24T19:41:38.677	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T19:58:59.833	NaN	2011-03-24T19:58:59.833	2011-03-24T19:58:59.833	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T20:05:07.753	NaN	2011-03-24T20:05:07.753	2011-03-24T20:05:07.753	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False
NaN	2011-03-24T20:22:44.603	NaN	2011-03-24T20:22:44.603	2011-03-24T20:22:44.603	...	NaN	5	0	NaN	NaN	NaN	NaN	0	[]	False

Рис. 6.4. Визуализация нескольких строк данных

После того как вы убедитесь, что данные соответствуют ожиданиям, изложенным в документации датасета, можно будет начать необходимую для моделирования обработку. Прежде всего это включает в себя очистку данных.

Очистка данных и отбор признаков

Следующий шаг в большинстве пайплайнов — удалить ненужную информацию, например поля или значения, которые не будут использоваться моделью, а также любые поля с информацией, недоступной модели в эксплуатационном окружении (см. раздел «Разделение датасета» на с. 128).

Помните о том, что каждый удаляемый вами признак теоретически может быть предиктором вашей модели. Принятие решения о том, какие признаки следует удалить, а какие — оставить, называется *отбором признаков* и является неотъемлемой частью процесса доработки моделей.

При этом нужно проследить за тем, чтобы не была потеряна какая-либо важная информация, были удалены все ненужные значения и в датасете не осталось никакой дополнительной информации, способной привести к искусственному завышению производительности модели за счет утечки данных (см. раздел «Утечка данных» на с. 131).

После очистки данных вам потребуется создать несколько полезных для модели признаков.

Генерация признаков

При введении нового признака, например частоты упоминания названия продукта в описании на Kickstarter, важно проверять принимаемые этим признаком значения. Необходимо следить за тем, чтобы признак всегда был заполнен и его значение было разумным. Это достаточно сложная задача, поскольку вам нужно определить не только все признаки, но и диапазон допустимых значений для каждого из них.

При этом пока не нужно слишком углубляться в анализ, поскольку на данном этапе вы должны лишь проверить правильность своих предположений о поступающих в модель данных, не касаясь вопроса о полезности данных или модели.

После создания признаков нужно будет убедиться в том, что вы сможете передать их модели в понятном для нее формате.

Форматирование данных

Как уже говорилось в предыдущих главах, перед тем как передавать элементы данных модели, их необходимо преобразовать в понятный для нее формат. Этот шаг может включать, например, нормализацию входных значений, векторизацию текста (его представление в числовом виде) или преобразование черно-белого видео в трехмерный тензор (см. раздел «Векторизация» на с. 95).

Если вы проводите обучение с учителем, то входные данные будут дополняться метками, такими как название класса в случае классификации или сегментационная карта в случае сегментации изображений. Эти метки также нужно будет преобразовать в понятный для модели формат.

Например, как показывает мой опыт решения задач сегментации изображений, причиной ошибок очень часто является несоответствие между метками и предсказаниями модели. Сегментационные модели используют в качестве меток сегментационные маски. Эти маски имеют такой же размер, как и входное изображение, но вместо пиксельных значений содержат метки класса для каждого пикселя. К сожалению, в разных библиотеках используются разные соглашения в отношении способа представления этих масок, из-за чего метки часто оказываются в неподходящем формате, что не позволяет модели обучаться.

Пример такой ошибки показан на рис. 6.5. Допустим, модель рассчитывает на то, что сегментационные маски будут передаваться со значением 255 в том случае, когда пиксель относится к определенному классу, и 0 в противном случае. Если пользователь при этом предполагает, что относящимся к классу пикселям должно присваиваться значение 1, а не значение 255, то он будет передавать свои маски модели в формате как на изображении «Предоставленные маски». Это приведет к тому, что модель посчитает такие маски практически пустыми и будет выдавать в итоге неточные предсказания.

Сходным образом в случае классификации метки часто представляют собой список из множества нулей и одной единицы, индекс которой соответствует некоторому реальному классу. Простая ошибка на одну позицию при этом приведет к смещению всех меток, в результате чего модель научится всегда предсказывать метки со смещением на одну позицию. Подобные ошибки трудно устранить, если не уделить время изучению данных.

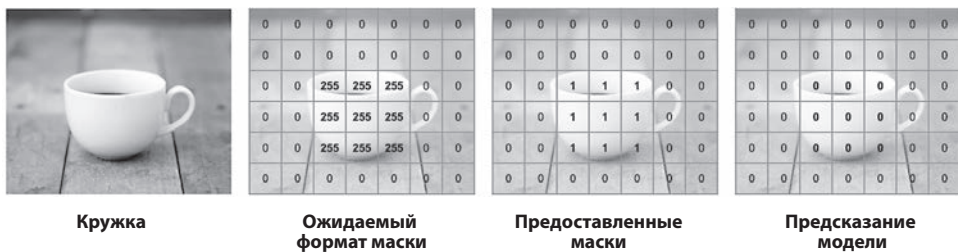


Рис. 6.5. При использовании неправильно отформатированных меток модель не сможет обучаться

МО-модели стараются подстроиться под большинство числовых входных данных вне зависимости от того, насколько они корректны в плане структуры

или содержания. Таким образом, на этом этапе часто происходит много трудноуловимых ошибок, в устранении которых очень помогает проверка формата.

Ниже показано, как может выглядеть функция проверки форматирования для нашего учебного примера. Сначала мы векторизуем текст вопроса, после чего добавляем к нему дополнительные признаки. Поскольку функция включает в себя несколько преобразований и векторных операций, визуализация возвращаемого ею значения позволяет нам убедиться в том, что она форматирует данные нужным нам образом.

```
def get_feature_vector_and_label(df, feature_names):
    """
    Генерируем входные и выходные векторы, используя векторные признаки
    и заданные названия признаков
    :param df: входной датафрейм
    :param feature_names: название столбцов, содержащих признаки
    (помимо векторов)
    :return: массив признаков и массив меток
    """
    vec_features = vstack(df["vectors"])
    num_features = df[feature_names].astype(float)
    features = hstack([vec_features, num_features])
    labels = df["Score"] > df["Score"].median()
    return features, labels

features = [
    "action_verb_full",
    "question_mark_full",
    "text_len",
    "language_question",
]

X_train, y_train = get_feature_vector_and_label(train_df, features)
```

Форматирование данных для модели, как правило, включает в себя множество операций — это в особенности характерно для текстовых данных. Преобразование строки текста в токенизированный список, а затем в векторизованное представление, часто включающее в себя дополнительные признаки, — процесс, чреватый ошибками. В силу этого даже обычная проверка размерности объектов на каждом этапе может выявить простые ошибки.

Как только данные будут в надлежащем формате, их можно передать модели. Последний шаг состоит в визуализации и валидации результатов модели.

Вывод модели

Прежде всего, изучение выходных данных позволяет убедиться в том, что модель выдает предсказания надлежащего типа и размерности (например, если мы пытаемся предсказать, сколько стоит объект недвижимости и как давно он выставлен на продажу, то модель должна выдавать массив из двух чисел).

Кроме того, когда для обучения модели используется только пара образцов, вы должны отслеживать, приближаются ли результаты к реальным меткам. Если модель не подстраивается под образцы, это может говорить о том, что данные были неправильно отформатированы или искажены.

Если по мере обучения результаты модели вообще не меняются, возможно, модель в действительности не задействует входные данные. В таком случае я рекомендую обратиться к разделу «Используйте опыт предшественников» на с. 55 и проверить, правильно ли применяется модель.

После того как вы прогоните через весь пайплайн несколько образцов, можно будет написать ряд тестов, чтобы автоматизировать часть работы по визуализации.

Организуем визуальную валидацию

Описанная выше работа по визуализации позволяет выявить значительную часть ошибок, так что посвятить этому время при создании пайплайна — правильное решение. Проверка предположений о том, как данные должны проходить через модель, позволяет сэкономить время, которое можно будет потратить на обучение и обобщение.

В то же время надо сказать, что пайплайны подвержены частым изменениям. Как гарантировать, что все будет работать так же хорошо, если вы будете вновь и вновь улучшать модель и менять алгоритм обработки? Если каждый раз прогонять через пайплайн образец с визуализацией его на всех этапах, это быстро станет слишком утомительным.

Именно здесь вступают в игру передовые практики программной разработки, о которых мы упоминали ранее. Пришла пора изолировать каждую часть пайплайна и закодировать наши наблюдения в виде тестов, которые можно будет запускать для проверки пайплайна по мере его изменения.

Разделение задач

Как и в случае обычной программной разработки, в МО очень полезно использовать модульную организацию. Чтобы упростить отладку и сейчас и в будущем, разделите функции так, чтобы запускать и проверять их можно было по отдельности, а не сразу весь пайплайн.

После того как вы разобьете пайплайн на отдельные функции, можно будет написать тесты для каждой из них.

Тестирование МО-кода

Тестирование модели — непростая задача. В то же время значительная часть кода пайплайна МО не имеет отношения к пайплайну обучения или самой

модели. Если вы еще раз взглянете на пайплайн из раздела «Начинайте с простого пайплайна» на с. 61, то увидите, что большинство функций там обладает детерминированным поведением и хорошо поддается тестированию.

Как показывает мой опыт сотрудничества с инженерами и специалистами по обработке данных, подавляющее большинство ошибок возникает из-за способа получения, обработки и передачи данных в модель. Поэтому для создания успешного МО-продукта очень важно протестировать логику обработки данных.

Для получения более подробных сведений о том, какие тесты можно использовать для МО-системы, я рекомендую ознакомиться со статьей Э. Брека (E. Breck) и др. «Тестовая оценка МО: критерий для оценки готовности МО к эксплуатации и снижения технической задолженности» («The ML Test Score: A Rubric for ML Production Readiness and Technical Debt Reduction»¹), где приводится много примеров и выводов, сделанных на основе опыта развертывания таких систем в компании Google.

В следующем разделе мы рассмотрим тесты, которые обычно используются для трех ключевых областей. На рис. 6.6 показано, что это за области и какие именно тесты применяются для каждой из них.

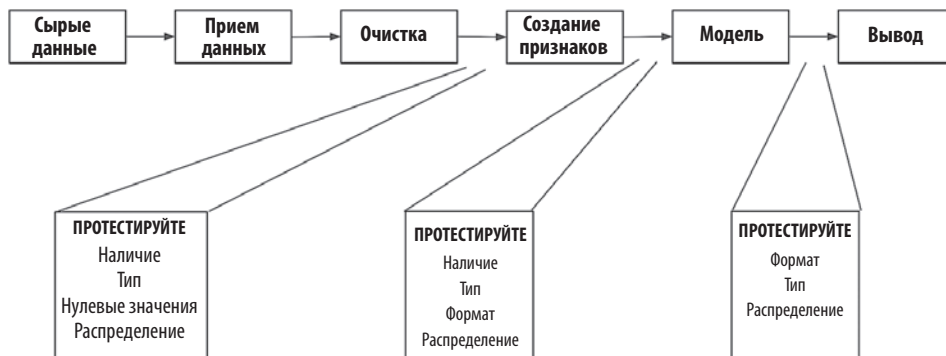


Рис. 6.6. Три ключевые области проверки

Поскольку работа любого пайплайна начинается с приема данных, именно с этой области мы и начнем свое тестирование.

Тестирование приема данных

Обычно данные хранятся в сериализованном виде на диске или в базе данных. При перемещении данных из хранилища в пайплайн мы должны проверять их

¹ <https://oreil.ly/OjYVl>

на предмет целостности и корректности. Для начала можно написать тесты, чтобы проверить наличие всех необходимых признаков в загружаемых данных.

Ниже представлены три теста, которые проверяют, что наш парсер возвращает правильный тип данных (датафрейм), что в нем есть все необходимые столбцы и ни один из столбцов не состоит только из нулевых значений. Все эти тесты (и некоторые другие) можно найти в папке тестов в GitHub-репозитории книги.

```
def test_parser_returns_dataframe():
    """
    Проверяет, запускается ли наш парсер
    и возвращает ли он DataFrame
    """
    df = get_fixture_df()
    assert isinstance(df, pd.DataFrame)

def test_feature_columns_exist():
    """
    Проверяет наличие всех необходимых столбцов
    """
    df = get_fixture_df()
    for col in REQUIRED_COLUMNS:
        assert col in df.columns

def test_features_not_all_null():
    """
    Проверяет отсутствие столбцов, содержащих только нулевые значения
    """
    df = get_fixture_df()
    for col in REQUIRED_COLUMNS:
        assert not df[col].isnull().all()
```

Мы также можем проверять тип каждого признака и то, что его значение не нулевое. Наконец, мы можем закодировать свои предположения о распределении и диапазоне значений каждого признака, а также вычислить их среднее, минимальное и максимальное значения. Например, недавно появившаяся библиотека Great Expectations¹ позволяет сразу проверить распределения признаков.

Вот как может выглядеть простейший тест для проверки среднего значения:

```
ACCEPTABLE_TEXT_LENGTH_MEANS = pd.Interval(left=20, right=2000)

def test_text_mean():
    """
    Проверяет, соответствует ли ожиданиям средняя длина текста
    """
    df = get_fixture_df()
```

¹ <https://oreil.ly/VG6b1>

```
df["text_len"] = df["body_text"].str.len()
text_col_mean = df["text_len"].mean()
assert text_col_mean in ACCEPTABLE_TEXT_LENGTH_MEANS
```

Эти тесты позволяют проследить за тем, чтобы вне зависимости от изменений, происходящих на стороне хранилища или API источника данных, модель имела доступ к таким же данным, на каких обучалась. После того как вы убедитесь в корректности принимаемых данных, можно будет перейти к следующему этапу пайплайна — обработке данных.

Тестирование обработки данных

Убедившись в том, что поступающие в пайплайн данные соответствуют нашим ожиданиям, мы также должны убедиться в надлежащей работе этапов очистки данных и создания признаков. Для начала можно написать тесты для проверки того, работает ли функция предобработки так, как мы ожидали. В дополнение к этому можно написать такие же тесты, как и для приема данных, чтобы понимать, верны ли наши предположения о состоянии поступающих в модель данных.

Таким образом, мы должны проверить наличие, тип и характеристики элементов данных на выходе из пайплайна обработки. Ниже показано, как могут выглядеть тесты для такой проверки, а также минимального, максимального и среднего значений:

```
def test_feature_presence(df_with_features):
    for feat in REQUIRED_FEATURES:
        assert feat in df_with_features.columns

def test_feature_type(df_with_features):
    assert df_with_features["is_question"].dtype == bool
    assert df_with_features["action_verb_full"].dtype == bool
    assert df_with_features["language_question"].dtype == bool
    assert df_with_features["question_mark_full"].dtype == bool
    assert df_with_features["norm_text_len"].dtype == float
    assert df_with_features["vectors"].dtype == list

def test_normalized_text_length(df_with_features):
    normalized_mean = df_with_features["norm_text_len"].mean()
    normalized_max = df_with_features["norm_text_len"].max()
    normalized_min = df_with_features["norm_text_len"].min()
    assert normalized_mean in pd.Interval(left=-1, right=1)
    assert normalized_max in pd.Interval(left=-1, right=1)
    assert normalized_min in pd.Interval(left=-1, right=1)
```

Эти тесты позволяют оценить влияние вносимых в пайплайн изменений на входные данные модели без необходимости писать какие-либо дополнительные тесты. Новые тесты потребуются только при добавлении новых признаков или при изменении входных данных.

После того как вы убедитесь в корректности принимаемых данных и их предобработки, можно будет перейти к проверке следующей части пайплайна — модели.

Тестирование вывода модели

Нам потребуется написать тесты для проверки корректности, размерности и диапазонов значений выходных данных модели. Также следует выполнять проверку предсказаний для ряда конкретных образцов. Это позволит заранее выявить снижение качества предсказаний новых моделей и проследить за тем, чтобы любая используемая нами модель всегда выдавала для этих образцов ожидаемый вывод. Когда новая модель в целом демонстрирует более высокую производительность, бывает трудно установить, не снижается ли ее производительность на каких-либо типах входных данных. Написание тестов существенно упрощает выявление подобных проблем.

В представленном ниже коде сначала проверяется формат и значения выдаваемых моделью предсказаний. Третий тест призван исключить снижение качества предсказаний, гарантируя, что модель классифицирует конкретный плохо сформулированный вопрос как низкокачественный.

```
def test_model_prediction_dimensions(
    df_with_features, trained_v1_vectorizer, trained_v1_model
):
    df_with_features["vectors"] = get_vectorized_series(
        df_with_features["full_text"].copy(), trained_v1_vectorizer
    )
    features, labels = get_feature_vector_and_label(
        df_with_features, FEATURE_NAMES
    )

    probas = trained_v1_model.predict_proba(features)
    # Модель делает одно предсказание для каждого входного образца
    assert probas.shape[0] == features.shape[0]
    # Модель предсказывает вероятности для двух классов
    assert probas.shape[1] == 2

def test_model_proba_values(
    df_with_features, trained_v1_vectorizer, trained_v1_model
):
    df_with_features["vectors"] = get_vectorized_series(
        df_with_features["full_text"].copy(), trained_v1_vectorizer
    )

    features, labels = get_feature_vector_and_label(
        df_with_features, FEATURE_NAMES
    )

    probas = trained_v1_model.predict_proba(features)
    # Модель выдает вероятности в диапазоне от 0 до 1
    assert (probas >= 0).all() and (probas <= 1).all()
```

```
def test_model_predicts_no_on_bad_question():
    input_text = "This isn't even a question. We should score it poorly"
    is_question_good = get_model_predictions_for_input_texts([input_text])
    # Модель классифицирует вопрос как плохо сформулированный
    assert not is_question_good[0]
```

Таким образом, мы начали с визуализации данных для проверки того, что они остаются полезными и пригодными для использования на всем пайплайне. Затем мы написали тесты, гарантирующие, что эти предположения останутся верными по мере изменения нашей стратегии обработки. Пришла пора заняться второй частью схемы, представленной на рис. 6.2, то есть отладкой обучения.

Отладка обучения: заставьте модель учиться

После того как вы проверите свой пайплайн и убедитесь, что он работает для одного образца, вы будете уверены в следующем: ваш пайплайн успешно принимает и преобразует данные; он передает данные модели в необходимом формате; ваша модель способна принять несколько элементов данных, обучиться на них и выдать корректные результаты.

Пришло время посмотреть, можете ли вы обучить модель на большом количестве образцов и *заставить ее подстроиться под все ваши обучающие данные*.

Для этого необходимо передать модели весь обучающий набор и проверить производительность. При использовании большого датасета можно избрать другой подход: постепенно увеличивайте количество передаваемых модели данных, все время отслеживая общую производительность.

Одно из преимуществ постепенного увеличения обучающего датасета состоит в том, что такой подход позволяет оценить влияние дополнительных данных на производительность модели. Передайте модели сначала несколько сотен образцов, затем несколько тысяч и, наконец, весь датасет (если же в нем меньше тысячи образцов, можно спокойно пропустить промежуточный шаг).

На каждом шаге обучайте модель и оценивайте ее производительность *на тех же данных*. Если модель способна обучаться на используемых вами данных, то ее производительность на обучающих данных должна оставаться сравнительно постоянной.

Чтобы посмотреть на производительность модели в реальной обстановке, я рекомендую оценить приемлемую ошибку для решаемой задачи. Это можно сделать, к примеру, вручную разметив несколько образцов и сравнив полученные предсказания с реальными метками. У большинства задач имеется неустранимая ошибка, то есть наилучшее значение производительности с учетом сложности задачи. На рис. 6.7 показано, как обычно выглядит график производительности обучения при использовании таких показателей.

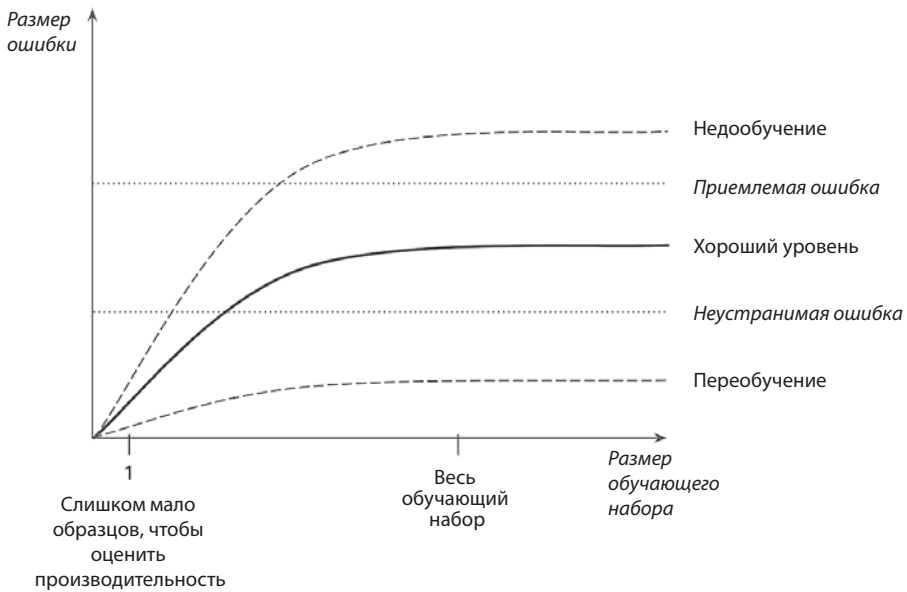


Рис. 6.7. Зависимость точности обучения от размера датасета

На всем датасете модель должна демонстрировать меньшую производительность, чем при использовании только одного образца, поскольку запомнить весь обучающий датасет труднее, чем один образец. В то же время производительность не должна выходить за определенные ранее границы.

Если вам удастся передать модели весь обучающий набор и получить производительность в границах, определенных вами при изучении цели продукта, то можно смело переходить к следующему этапу. Тем не менее в следующем разделе я опишу несколько типичных причин, по которым модель может испытывать трудности на обучающем наборе.

Сложность задачи

Если производительность модели существенно ниже, чем ожидалось, это может объясняться слишком высокой сложностью задачи. Чтобы оценить сложность задачи, проверьте следующее:

- количество и разнообразие используемых вами данных;
- насколько прогностически значимыми являются созданные вами признаки;
- сложность модели.

Давайте чуть подробнее остановимся на каждом из этих аспектов.

Качество, количество и разнообразие данных

Чем более разносторонней и сложной является решаемая задача, тем больше данных требуется использовать для обучения модели. Чтобы модель могла «выучить» имеющиеся закономерности, нужно предоставить ей множество образцов разных категорий. Так, например, если вам нужно относить изображения кошек к одной из ста пород, вам потребуется гораздо больше изображений, чем в том случае, когда нужно просто отличать кошек от собак. На самом деле при возрастании количества классов количество данных часто возрастает экспоненциально, поскольку с ростом количества классов растет и вероятность неправильной классификации.

Также следует отметить, что чем меньше у вас данных, тем сильнее будет влияние ошибок в метках или отсутствующих значений. Поэтому советую потратить какое-то время на изучение и проверку признаков и меток используемого датасета.

Наконец, в большинстве датасетов имеются *выбросы* — элементы данных, которые сильно отличаются от других, в силу чего модели сложно с ними работать. Удаление таких выбросов из обучающего набора часто позволяет повысить производительность модели и упростить задачу. В то же время этот подход не всегда уместен — если предполагается, что модель будет сталкиваться с аналогичными элементами данных при эксплуатации, будет лучше оставить выбросы и *сосредоточиться на улучшении данных и модели*, чтобы модель смогла успешно подстроиться под данные.

Чем сложнее используемый датасет, тем полезнее будет поработать над способом представления данных, чтобы модели было проще обучаться. Давайте посмотрим, что это означает.

Способ представления данных

Насколько легко выявить необходимые закономерности при использовании текущего способа представления данных? Если модель отказывается хорошо работать на обучающих данных, вы должны добавить признаки, которые сделают данные более выразительными, что позволит модели обучаться эффективнее.

Это могут быть прогностически значимые признаки, которые мы ранее игнорировали. Первая версия модели для нашего МО-редактора принимала в расчет только текст в теле вопроса. Однако более тщательное изучение датасета показало, что заголовки вопросов также очень информативны. Снова включив этот признак в датасет, мы смогли повысить производительность модели.

Новые признаки также могут создаваться путем доработки существующих признаков или их творческого комбинирования. Пример такого подхода уже приводился в разделе «Используйте данные для принятия решений о признаках и моделях» на с. 113, где нам нужно было каким-то образом скомбинировать

день недели и день месяца, чтобы получить признак, актуальный для конкретной бизнес-задачи.

Иногда проблема заключается непосредственно в модели. Давайте остановимся на таких случаях подробнее.

Производительность модели

Наибольшую пользу чаще всего приносит повышение качества данных и улучшение признаков. Когда причина низкой производительности заключается в самой модели, это часто объясняется тем, что она не подходит для решаемой задачи. Как было показано в разделе «От закономерностей к моделям» на с. 126, различные датасеты и задачи требуют использования различных типов моделей. Неподходящая для задачи модель не сможет продемонстрировать высокую производительность, даже если ей удалось переобучиться на нескольких образцах.

Если модель показывает плохие результаты на датасете, который, как вам кажется, имеет много прогностически значимых признаков, сначала спросите себя, используете ли вы подходящую модель. Если это возможно, используйте более простую версию вашей модели, чтобы было проще ее изучить. Например, если модель типа «случайный лес» отказывается показывать сколько-нибудь хорошие результаты, попробуйте использовать на той же задаче дерево решений и визуализируйте отдельные подмножества данных, чтобы выяснить, использует ли модель те признаки, которые, по вашему мнению, являются прогностически значимыми.

Еще одной причиной проблем может быть излишняя простота используемой модели. Хотя использовать максимально простую модель в самом начале полезно, поскольку это позволяет быстро ее дорабатывать, следует иметь в виду, что некоторые модели совершенно не способны справиться с определенными типами задач. Для таких задач часто требуется усложнить модель. Чтобы проверить, насколько хорошо модель адаптирована к решаемой задаче, я рекомендую изучить имеющиеся наработки в рассматриваемой области, как было описано в разделе «Используйте опыт предшественников» на с. 55. Найдите примеры аналогичных задач и посмотрите, какие модели использовались для их решения. Одна из этих моделей может стать хорошей отправной точкой.

Если вы видите, что модель подходит для решаемой задачи, то причина ее низкой производительности, вероятно, заключается в обучении.

Проблемы оптимизации

Мы начали с проверки того, способна ли модель подстроиться под небольшой набор образцов. Поэтому мы уверены, что данные смогут поступать в прямом

и обратном направлении. Однако нам неизвестно, сможет ли модель подстроиться под весь датасет. Используемый ею метод модификации весов может не подходить для текущего датасета. Такие проблемы очень характерны для достаточно сложных моделей наподобие нейронных сетей, производительность которых часто сильно зависит от выбора гиперпараметров.

При работе с моделями, которые, подобно нейронным сетям, обучаются, используя градиентный спуск, выявить проблемы обучения часто помогает применение таких инструментов визуализации, как дашборд TensorBoard¹. Построив график потерь в процессе оптимизации, вы увидите, что уровень потерь сначала снижается резко, а затем более плавно. На рис. 6.8 показано, как может выглядеть график потерь в процессе обучения на панели TensorBoard — в данном случае он обозначен как «кросс-энтропия» (cross entropy).

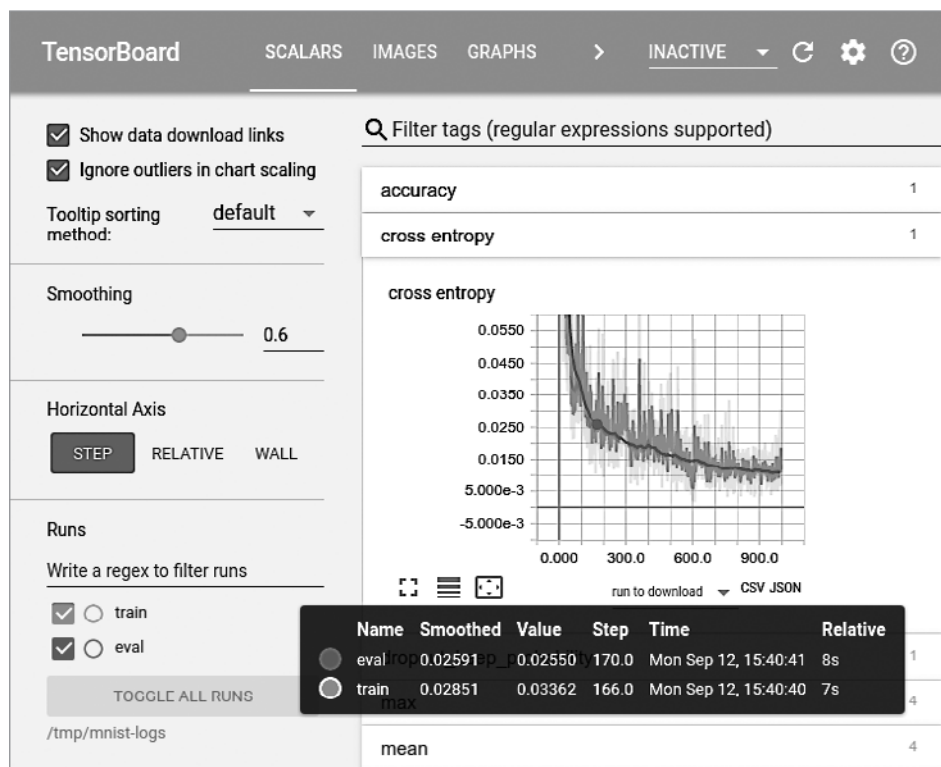


Рис. 6.8. Дашборд TensorBoard; скриншот из документации по TensorBoard

¹ <https://oreil.ly/xn2tY>

Иногда эта кривая демонстрирует очень медленное уменьшение потерь, что может быть признаком слишком медленного обучения модели. В таком случае можно повысить скорость обучения и посмотреть, возросла ли скорость уменьшения потерь, построив кривую еще раз. Если же изменение потерь происходит нестабильно, это может быть признаком слишком быстрого обучения модели.

Наряду с визуализацией потерь нужно посмотреть, насколько хорошо обучается нейронная сеть. Это можно сделать с помощью визуализации весов и активации. На рис. 6.9 показано, как может изменяться распределение весов по мере обучения модели. Если распределение весов почти не изменяется в течение нескольких эпох, это может говорить о том, что нужно повысить скорость обучения. Если же распределение весов с каждым разом изменяется слишком сильно, то вам нужно снизить скорость обучения.

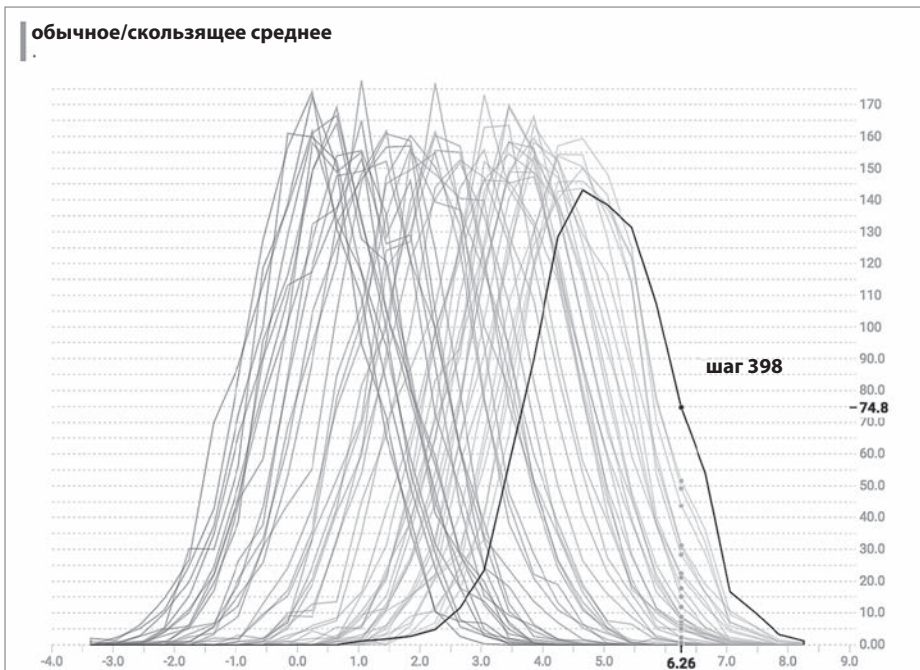


Рис. 6.9. Гистограмма изменения весов по мере обучения

То, что модель подстроилась под обучающие данные, — важное достижение в ходе работы над проектом МО, однако это еще не финальная точка. Конечной целью при создании МО-продукта является создание модели, способной хорошо работать на незнакомых ей образцах. Поэтому нам нужна модель, способная уверенно обобщать незнакомые данные, что мы и рассмотрим далее.

Отладка обобщения: модель должна быть полезной

Обобщение — третья и заключительная часть схемы, представленной на рис. 6.2; этот процесс призван обеспечить хорошую работу модели на незнакомых ей данных. В разделе «Разделение датасета» на с. 128 было показано, что для оценки способности модели обобщать незнакомые ей данные важно создать отдельные выборки для обучения, валидации и тестирования. В разделе «Оценка модели: не ограничивайтесь точностью» на с. 140 мы рассмотрели методы оценки производительности модели и выявления дополнительных признаков, которые теоретически могут обеспечить более эффективную работу модели. В данном разделе будет приведен ряд рекомендаций на тот случай, когда модель отказывается хорошо работать на валидационном наборе после выполнения нескольких итераций.

Утечка данных

Хотя мы уже говорили об утечке данных в разделе «Утечка данных» на с. 131, будет нелишним еще раз коснуться этой темы в контексте обобщения. Модель часто изначально работает на валидационном наборе хуже, чем на обучающем. Это не должно удивлять, поскольку модели труднее делать предсказания на незнакомых данных.



Оценка потерь на валидационных и обучающих данных в ходе обучения может показывать, что производительность на валидационных данных выше, чем на обучающих. Это объясняется тем, что в ходе обучения на тренировочной выборке потери накапливаются на протяжении всей эпохи, а потери на валидационном наборе рассчитываются уже после завершения эпохи, с использованием последней версии модели.

Если производительность на валидационном наборе выше, чем на обучающем, это может быть признаком утечки данных. Если образцы из обучающего набора будут содержать информацию об образцах из валидационного набора, то модель сможет использовать эту информацию и покажет хорошие результаты на валидационном наборе. Если вас удивляет высокая производительность на валидационном наборе, посмотрите, какие признаки использует модель и не происходит ли при этом утечки данных. Устранение утечки приведет к ухудшению результатов на валидационном наборе, зато позволит получить более эффективную модель.

Таким образом, при наличии утечки данных создается ошибочное впечатление, что модель хорошо обобщает данные, в то время как в действительности она на это не способна. В то же время оценка производительности на резервном

валидационном наборе может показать, что модель хорошо работает только на обучающих данных. В таких случаях не исключена вероятность того, что модель была переобучена.

Переобучение

В разделе «Дилемма смещения-дисперсии» на с. 138 мы узнали, что в случае, когда модель плохо подогнана под обучающие данные, говорят, что модель «недообучена», а когда модель подстраивается *слишком хорошо* — «переобучена».

Что же это значит? Это значит, что вместо того, чтобы выучить обобщаемые тренды, которые, к примеру, коррелируют с высоким или низким качеством текста, модель может выявить специфические закономерности, которые присутствуют в отдельных образцах обучающего набора и отсутствуют в других. Эти закономерности позволяют модели показать хорошие результаты на обучающем наборе, но оказываются бесполезными при классификации других образцов.

На рис. 6.10 показано, как могут на практике выглядеть ситуации переобучения и недообучения при использовании небольшого датасета. Переобученная модель прекрасно подогнана под обучающие данные, но неточно аппроксимирует общий тренд; как следствие, она не может точно предсказать незнакомые точки. Недообученная модель вообще не извлекает имеющийся в данных тренд. По сравнению с переобученной моделью умеренно подогнанная модель демонстрирует более низкую производительность на обучающих данных, но лучше справляется с незнакомыми данными.

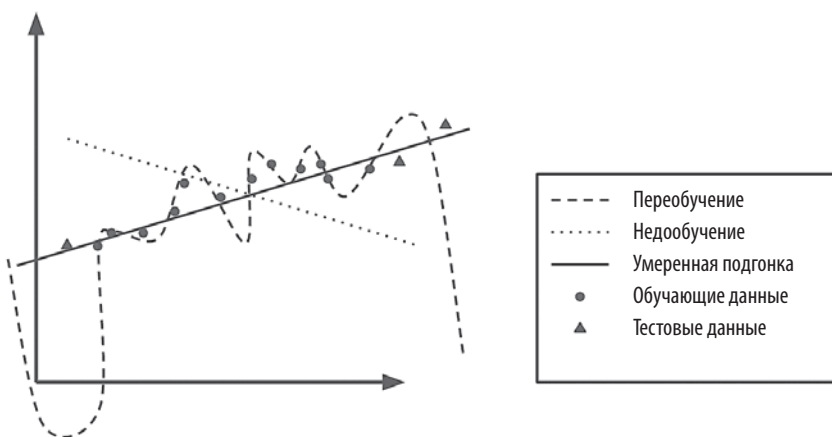


Рис. 6.10. Переобучение и недообучение

Когда модель работает на обучающем наборе гораздо лучше, чем на тестовом, это обычно означает, что она переобучена. Она выучила специфические особенности обучающих данных, но не способна хорошо работать на незнакомых данных.

Поскольку к переобучению ведет извлечение моделью слишком большого количества информации из обучающих данных, мы можем избежать этого, ограничив обучающую способность модели. Существует несколько способов это сделать; давайте рассмотрим каждый из них.

Регуляризация

Регуляризация (Regularization) ограничивает способность модели представлять информацию. Она смещает внимание модели с множества малополезных признаков на выбор меньшего количества более значимых признаков.

Широко используемый способ регуляризации модели — наложение штрафа на абсолютную величину ее весов. Так, например, методы L1- и L2-регуляризации, предназначенные для линейной и логистической регрессии, добавляют в функцию потерь дополнительное условие, которое накладывает штраф на большие веса. В случае L1-регуляризации это условие — сумма абсолютных величин весов, а в случае L2-регуляризации — сумма квадратов весов.

Применение разных методов регуляризации производит разный эффект. L1-регуляризация часто упрощает выбор информативных признаков за счет того, что неинформативные признаки приравниваются к нулю (подробности см. в статье Википедии, посвященной методу регрессии «лассо»¹). Еще один полезный эффект L1-регуляризации состоит в том, что при наличии взаимосвязанных признаков она побуждает модель использовать только один из них.

Методы регуляризации также могут быть ориентированными на конкретные виды моделей. Так, в качестве метода регуляризации нейронных сетей часто используется метод отсева (dropout). Этот метод случайным образом игнорирует некоторую часть нейронов сети на этапе обучения. Это не дает отдельным нейронам оказывать чрезмерное влияние, тем самым сеть не запоминает все особенности обучающих данных.

Что касается моделей на основе деревьев, таких как «случайный лес», ограничение максимальной глубины деревьев снижает вероятность чрезмерной подгонки отдельных деревьев под данные и, таким образом, помогает регуляризовать модель. Регуляризации также способствует увеличение количества используемых в «случайном лесу» деревьев.

Не допустить чрезмерной подгонки модели под обучающие данные также можно путем внесения в сами данные изменений, снижающих вероятность переобу-

¹ <https://oreil.ly/Su9Bf>

чения. Это можно сделать с помощью процесса, называемого *приращением данных* (или *аугментацией*).

Аугментация

Аугментация подразумевает создание новых обучающих данных путем незначительной модификации имеющихся элементов данных. Цель при этом состоит в том, чтобы искусственным образом получить элементы данных, отличающиеся от уже имеющихся, и предоставить модели более разнообразные входные данные. Стратегия приращения зависит от типа используемых данных. На рис. 6.11 показано несколько способов аугментации датасета с изображениями.

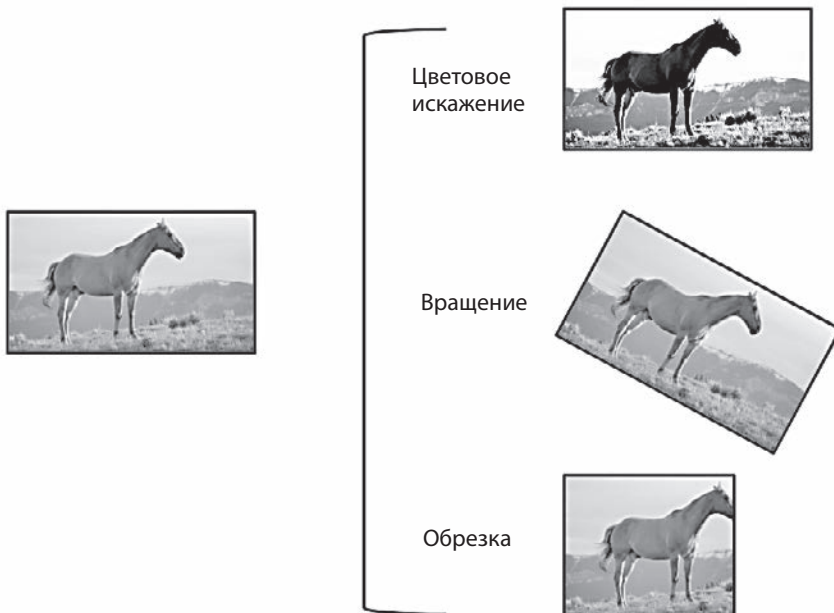


Рис. 6.11. Несколько способов аугментации датасета с изображениями

Аугментация делает обучающий набор менее однообразным и, следовательно, более сложным. Так модели труднее подстроиться под обучающие данные, но зато она получает более обширный диапазон входных данных на этапе обучения. Соответственно, аугментация данных часто ведет к снижению производительности на обучающем наборе, но зато повышает ее на незнакомых данных, например на валидационном наборе или при эксплуатации. Эта стратегия особенно эффективна в том случае, когда аугментация позволяет сделать данные обучающего набора более похожими на образцы, с которыми приходится иметь дело на практике.

Мне как-то довелось помогать инженеру, который с помощью спутниковых снимков пытался определить, какие дороги будут затоплены в случае урагана. Это была непростая задача, поскольку он располагал только размеченными данными по городам, которые не подвергались действию урагана. Чтобы повысить производительность своей модели на гораздо более темных и менее качественных снимках районов, подверженных действию урагана, он создал пайплайны аугментации данных, которые делали обучающие образцы изображений более темными и размытыми. Учитывая, что теперь на снимках было труднее различить дороги, производительность модели на обучающих данных снизилась, однако повысилась производительность на валидационном наборе, поскольку аугментация приблизила изображения к данным валидационного набора. Аугментация данных сделала обучающий набор более репрезентативным, что, в свою очередь, обеспечило более надежную работу модели.

Если производительность модели на валидационном наборе будет недостаточной даже после применения описанных выше методов, нужно будет доработать сам датасет.

Доработка датасета

В некоторых случаях трудности с разделением данных на обучающий и валидационный набор могут приводить к недообучению модели и низкой производительности на валидационном наборе. Если обучающий набор будет содержать только простые образцы, а валидационный набор — только сложные, то модель не сможет обучаться на сложных элементах данных. Сходным образом в обучающем наборе может быть недостаточно образцов определенной категории, что не позволит модели обучаться на них. Если целью обучения является оптимизация агрегатной метрики, не исключена вероятность того, что модель подстроится под большинство классов и будет игнорировать остальные.

Хотя здесь может помочь аугментация данных, наилучший подход часто состоит в том, чтобы доработать способ разделения данных, сделав более репрезентативным обучающий набор. При этом важно проследить, чтобы не было утечки данных и выделяемые наборы были максимально сбалансированными в плане сложности. Если новый способ разделения данных будет относить все простые образцы к валидационной выборке, то модель покажет на ней искусственно завышенную производительность, но не сможет показать столь же хорошие результаты в реальных условиях. Чтобы не беспокоиться, что наборы данных будут качественно отличаться, можно воспользоваться k -fold кросс-валидацией¹. Этот метод заключается в том, что вы k раз применяете различные способы разделения данных и оцениваете производительность модели на каждой паре наборов.

¹ <https://oreil.ly/NkhZa>

После того как вы сбалансируете обучающий и валидационный наборы таким образом, чтобы они практически не отличались по сложности данных, модель должна показать более высокую производительность. Если она по-прежнему будет недостаточной, это может объясняться чрезвычайно высокой сложностью задачи.

Изучение решаемой задачи

Модели может быть трудно обучиться из-за того, что решаемая задача слишком сложная. Иногда входные данные просто не позволяют предсказать целевой результат. Чтобы убедиться в том, что решаемая вами задача по своей сложности соответствует текущему уровню развития сферы МО, я рекомендую еще раз обратиться к разделу «Используйте опыт предшественников» на с. 55.

Кроме того, если у вас есть датасет, это еще не значит, что вашу задачу можно решить. Здесь в качестве примера можно привести невыполнимую задачу точного прогнозирования случайных выходных данных для случайных входных. Хотя вы можете создать модель, которая покажет высокую производительность на обучающем наборе, просто его запомнив, эта модель не сможет точно предсказывать случайные результаты для других случайных входных данных.

Если вашей модели не удастся производить обобщение, то вы, вероятно, взялись за слишком сложную задачу. Возможно, ваши обучающие образцы содержат слишком мало информации для того, чтобы можно было извлечь *значимые признаки*, дающие достаточное представление о тех элементах данных, с которыми вам придется иметь дело в дальнейшем. Если это так, то решаемая вами задача плохо подходит для МО, и я советую вам снова обратиться к главе 1, чтобы составить более удачную формулировку задачи.

Закключение

В этой главе вы узнали, какие три последовательных шага необходимо сделать, чтобы заставить модель работать. Прежде всего, надо отладить поток данных вашего пайплайна, изучив датасет и написав тесты. Затем следует добиться высокой производительности на обучающем наборе, чтобы убедиться, что модель способна обучаться. Наконец, удостовериться, что модель способна производить обобщение и выдавать полезные результаты на незнакомых данных.

Этот процесс поможет быстрее создать и отладить модели и обеспечит их надежность. Создав, обучив и отладив свою первую модель, вы должны оценить ее производительность и либо доработать ее, либо развернуть.

В главе 7 мы узнаем, как можно применить обученный классификатор для выдачи пользователям практически полезных рекомендаций. Затем мы сравним модели для нашего МО-редактора и решим, какую из них лучше использовать.

ГЛАВА 7

Использование классификаторов для выдачи рекомендаций

Лучший способ добиться прогресса в МО — многократное выполнение изображенного на рис. 7.1 цикла, который мы уже рассматривали во введении к части III. Сначала вы формулируете гипотезу моделирования, затем вновь и вновь дорабатываете пайплайн моделирования, потом проводите детальный анализ ошибок, чтобы на его основе сформулировать следующую гипотезу.



Рис. 7.1. Цикл МО

В предыдущих главах мы уже рассмотрели несколько этапов этого цикла. В главе 5 показали, как обучить модель и оценить ее в баллах. В главе 6 дали советы по ускорению моделирования и устранению ошибок, возникающих в процессе МО. Теперь мы рассмотрим последний пункт этого итерационного цикла. Нач-

нем с изучения способов применения обученного классификатора для выдачи пользователям рекомендаций, затем перейдем к выбору подходящей модели и наконец, объединив первое и второе, создадим работоспособный МО-редактор.

В разделе «Составление плана разработки МО-редактора» на с. 59 мы составили общий план, включающий в себя два этапа: обучение модели, способной классифицировать вопросы как вопросы с высоким или низким баллом, а затем использование обученной модели, которая помогает пользователям улучшить формулировку вопросов. Давайте посмотрим, как применить такую модель для выдачи советов по написанию текстов.

Вывод рекомендаций

Цель МО-редактора — давать рекомендации по написанию текстов. Классификация вопросов на хорошие и плохие — первый шаг, позволяющий оценить качество вопроса пользователя. Теперь мы должны сделать еще один шаг и помочь пользователю улучшить формулировку вопроса, предлагая практические рекомендации.

Этот раздел будет посвящен способам предоставления таких рекомендаций. Мы начнем с простых подходов, которые опираются на агрегатные метрики признаков и не требуют использования модели на этапе инференса. Затем посмотрим, как можно использовать и балл модели, и ее чувствительность к искажениям для создания более персонализированных рекомендаций. Примеры применения каждого из этих методов к МО-редактору можно найти в GitHub-репозитории книги, в ноутбуке, посвященном генерированию рекомендаций.

Чего мы можем добиться без модели?

Чтобы получить эффективную модель, надо выполнить несколько итераций в цикле МО. На каждой итерации создается все более удачный набор признаков за счет изучения имеющихся наработок в этой области, совершенствования датасетов и анализа результатов модели. Это итеративное улучшение признаков можно использовать и для предоставления пользователям рекомендаций. При таком подходе не нужно запускать модель для каждого вопроса пользователя, что позволяет больше сосредоточиться на общих рекомендациях.

При этом вы можете либо использовать признаки напрямую, либо задействовать обученную модель, чтобы легко отобрать наиболее актуальные.

Использование статистики по признакам

После выявления прогностически значимых признаков их можно передавать непосредственно пользователям, минуя модель. Если у разных классов суще-

ственно различаются средние значения признаков, то вы можете напрямую предоставить эту информацию, чтобы пользователи могли направить свои вопросы к верной категории.

Работая над МО-редактором, мы практически сразу выявили такой признак, как наличие вопросительных знаков. Изучение данных показало, что вопросы с высоким баллом, как правило, содержат меньше вопросительных знаков. Чтобы использовать эту информацию для выдачи рекомендаций, можно написать правило, которое будет предупреждать пользователя, что доля вопросительных знаков в его вопросе гораздо больше, чем в вопросах с высоким баллом.

Отобразить средние значения признаков для каждого класса можно с помощью библиотеки `pandas`. Достаточно написать всего несколько строк кода:

```
class_feature_values = feats_labels.groupby("label").mean()
class_feature_values = class_feature_values.round(3)
class_feature_values.transpose()
```

Результат выполнения данного кода показан в табл. 7.1. Как видите, значения многих из выделенных нами признаков сильно различаются для вопросов с высоким и низким баллом (метки «True» и «False» соответственно).

Таблица 7.1. Различия в значениях признаков классов

Метка	False	True
num_questions	0,432	0,409
num_periods	0,814	0,754
num_commas	0,673	0,728
num_exclam	0,019	0,015
num_quotes	0,216	0,199
num_colon	0,094	0,081
num_stops	10,537	10,610
num_semicolon	0,013	0,014
num_words	21,638	21,480
num_chars	822,104	967,032

Использование статистики по признакам является простейшим способом предоставления надежных рекомендаций и имеет много общего с тем эвристическим подходом, который мы изначально использовали в разделе «Простейший подход: “сам себе алгоритм”» на с. 37.

При сравнении значений признаков разных классов бывает трудно установить, какие признаки в наибольшей мере влияют на отнесение вопроса к тому или иному классу. Проанализировать это проще, если мы будем учитывать важность признаков.

Извлечение глобальной важности признаков

В разделе «Оценка важности признаков» на с. 152 мы уже говорили о том, как определить важность признаков в контексте оценки модели. Наряду с этим оценка важности признаков также может использоваться для ранжирования рекомендаций, выдаваемых на основе признаков. Для показа пользователю рекомендаций следует отдавать предпочтение тем признакам, которые являются наиболее прогностически значимыми для обученного классификатора.

Ниже показано, как выглядят результаты оценки важности признаков для модели классификации вопросов, использующей 30 признаков. Каждый из пяти наиболее важных признаков обладает более высокой степенью важности по сравнению с пятью наименее важными признаками, то есть с точки зрения модели пользователи смогут гораздо быстрее улучшить свои вопросы, если при выдаче им рекомендаций мы будем в первую очередь руководствоваться наиболее важными признаками.

Top 5 importances:

```
num_chars: 0.053
num_questions: 0.051
num_periods: 0.051
ADV: 0.049
ADJ: 0.049
```

Bottom 5 importances:

```
X: 0.011
num_semicolon: 0.0076
num_exclam: 0.0072
CONJ: 0
SCONJ: 0
```

Учитывая одновременно и статистику и важность признаков, можно сделать рекомендации более практическими и целенаправленными. При этом первый подход позволяет получить целевые значения каждого признака, а второй — отобразить пользователям лишь небольшое подмножество наиболее важных признаков. При этом рекомендации можно выдавать достаточно быстро, поскольку ни первый, ни второй подход не требуют запуска модели; достаточно лишь сверить входные данные со статистикой по наиболее важным признакам.

Как мы видели в разделе «Оценка важности признаков» на с. 152, если модель сложная, определить важность признаков бывает трудно. Если модель не позволяет извлечь важность признаков, можно использовать интерпретаторы «черного ящика» на большой выборке образцов.

Еще один недостаток статистики и важности признаков состоит в том, что они не всегда обеспечивают точные рекомендации. Поскольку статистическая информация собирается на основе всего датасета, основанные на ней рекомендации нельзя применить к каждому отдельному образцу. Статистика по признакам позволяет выдать лишь некоторые общие рекомендации, например сообщение о том, что вопросы с высоким баллом, как правило, содержат больше наречий. Однако иногда вопросы получают высокий балл, даже несмотря на то что содержат сравнительно мало наречий. В таких случаях эти рекомендации не принесут никакой пользы.

В следующих двух разделах мы узнаем, как выдавать более детальные рекомендации для отдельных образцов.

Использование балла модели

В главе 5 было показано, как классификаторы выдают балл для каждого образца. После этого образец относится к некоторому классу, если балл превышает определенное пороговое значение. Если балл модели хорошо откалиброван (тема калибровки была подробно рассмотрена в разделе «Калибровочная кривая» на с. 145), то его можно использовать в качестве оценки вероятности принадлежности образца к определенному классу.

Чтобы вместо класса отображался балл модели, вызовите функцию `predict_proba` библиотеки `scikit-learn`, указав класс.

```
# probabilities – это массив, содержащий по одной вероятности на класс
probabilities = clf.predict_proba(features)

# positive_probs содержит только балл положительного класса
positive_probs = clf[:,1]
```

Если балл хорошо откалиброван, то, видя его, пользователи смогут отслеживать улучшения в своем вопросе по мере выполнения рекомендаций. Такие механизмы быстрой обратной связи позволяют повысить доверие пользователей к выдаваемым рекомендациям.

Помимо откалиброванного балла, для выдачи рекомендаций по улучшению конкретного вопроса также можно использовать обученную модель.

Извлечение локальной важности признаков

Рекомендации для отдельного вопроса можно сгенерировать, используя интерпретатор «черного ящика» поверх обученной модели. В разделе «Оценка

важности признаков» на с. 152 мы узнали, что интерпретаторы «черного ящика» оценивают важность признаков для конкретного образца путем многократного внесения небольших изменений во входные признаки и наблюдения за тем, как это сказывается на выдаваемом моделью балле. Такой инструмент прекрасно подойдет для рекомендаций.

Чтобы убедиться в этом, давайте попробуем сгенерировать объяснения для образца с помощью пакета LIME¹. В примере кода ниже сначала создается табличный объект объяснения, после чего выбирается образец из наших тестовых данных. Затем мы отображаем объяснения в ноутбуке, посвященном генерированию рекомендаций, в GitHub-репозитории книги. Наконец, преобразуем объяснения в массив.

```
from lime.lime_tabular import LimeTabularExplainer

explainer = LimeTabularExplainer(
    train_df[features].values,
    feature_names=features,
    class_names=["low", "high"],
    discretize_continuous=True,
)

idx = 8
exp = explainer.explain_instance(
    test_df[features].iloc[idx, :],
    clf.predict_proba,
    num_features=10,
    labels=(1,),
)

print(exp_array)
exp.show_in_notebook(show_table=True, show_all=False)
exp_array = exp.as_list()
```

В результате выполнения кода мы получим график (рис. 7.2), а также массив с важностью признаков. Предсказанные моделью вероятности отображены слева на рисунке. В центре рисунка показано, как ранжируются значения признаков по их вкладу в предсказание.

Ниже, в консоли, представлены те же значения, что и на графике, но в более читабельном виде. Каждая строка в этом выводе представляет отдельное значение признака и его влияние на результат модели. Например, тот факт, что значение признака `num_diff_words` было меньше, чем 88,00, привел к снижению результата модели примерно на 0,038. Согласно данной модели, если длина вопроса будет выше этого числа, то качество улучшится.

¹ <https://github.com/marcotcr/lime>

```
[('num_diff_words <= 88.00', -0.038175093133182826),
 ('num_questions > 0.57', 0.022220445063244717),
 ('num_periods <= 0.50', 0.018064270196074716),
 ('ADJ <= 0.01', -0.01753028452563776),
 ('408.00 < num_chars <= 655.00', -0.01573650444507041),
 ('num_commas <= 0.39', -0.015551364531963608),
 ('0.00 < PROPEN <= 0.00', 0.011826217792851488),
 ('INTJ <= 0.00', 0.011302327527387477),
 ('CONJ <= 0.00', 0.0),
 ('SCONJ <= 0.00', 0.0)]
```

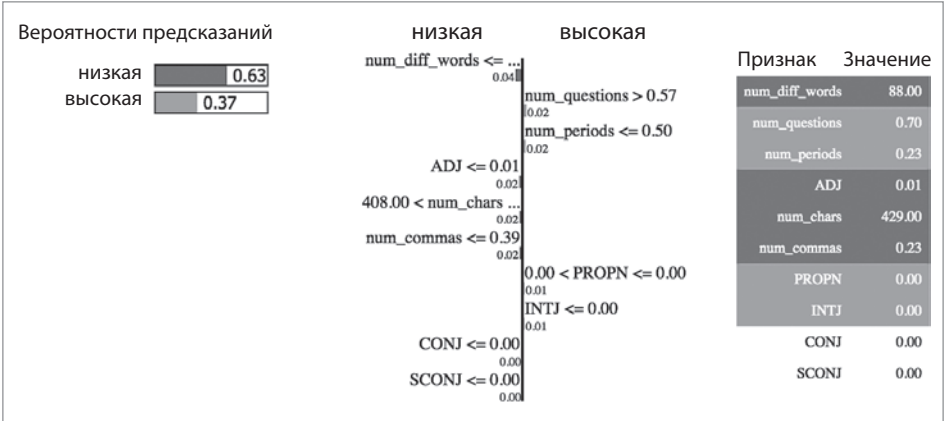


Рис. 7.2. Использование объяснений в качестве рекомендаций

В ноутбуке по генерированию рекомендаций в GitHub-репозитории книги вы можете найти еще несколько примеров.

Хотя интерпретаторы «черного ящика» позволяют генерировать точные рекомендации для отдельной модели, у них есть один недостаток. Поскольку ради оценки они изменяют признаки и для каждого образца запускают модель, выдача рекомендаций занимает больше времени по сравнению с описанными ранее методами. Так, например, для оценки важности признаков интерпретатор LIME по умолчанию производит 500 изменений. Это делает данный метод на два порядка медленнее тех методов, которые запускают модель один раз, и даже медленнее, чем те, которые вообще не требуют запуска модели. На моем компьютере запуск интерпретатора LIME для одного вопроса занимает чуть больше двух секунд. Такая задержка не позволит нам выдавать рекомендации пользователям по мере ввода текста; вместо этого они должны будут самостоятельно отправлять каждый вопрос.

Как и многие другие МО-модели, описанные методы выдачи рекомендаций являются компромиссом между точностью и задержкой. Какой из этих методов лучше подойдет для продукта, зависит от требований.

Все рассмотренные методы выдачи рекомендаций учитывают признаки, созданные на этапе доработки модели, а некоторые методы также используют обученные ранее модели. В следующем разделе мы сравним модели для МО-редактора и решим, какая из них лучше всего подходит для выдачи рекомендаций.

Сравнение моделей

В разделе «Оценка успешности» на с. 44 были рассмотрены важные показатели успешности продукта, а в разделе «Оценка производительности» на с. 137 мы описали методы оценки моделей. Эти методы также можно использовать для сравнения производительности последовательно создаваемых версий моделей и признаков.

В данном разделе мы отберем несколько ключевых метрик и используем их для оценки производительности и пользы трех последовательно созданных версий МО-редактора.

Для выдачи действенных рекомендаций модель должна удовлетворять следующим требованиям. Она должна быть хорошо откалибрована, чтобы выдаваемые вероятности давали информативную оценку качества вопроса. Как уже говорилось в разделе «Оценка успешности» на с. 44, модель должна иметь высокую точность (precision), чтобы мы могли выдавать верные рекомендации. Модель должна использовать понятные для пользователя признаки, поскольку рекомендации будут выдаваться на их основе. Наконец, чтобы можно было использовать интерпретатор «черного ящика», модель должна быть достаточно быстрой.

Давайте рассмотрим методы моделирования, которые последовательно использовались для МО-редактора, и сравним их производительность. Соответствующий код можно найти в ноутбук по сравнению моделей в GitHub-репозитории книги.

Версия 1: простейший отчет

В главе 3 мы создали первую версию редактора, основанную исключительно на эвристическом алгоритме. Эта версия использовала жестко заданные правила хорошей читабельности текста и отображала пользователям результаты в структурированном формате. Создание такого пайплайна позволило нам модифицировать свой подход и сосредоточиться в процессе МО на предоставлении более четких рекомендаций вместо простого набора показателей.

Поскольку этот исходный прототип был создан лишь для того, чтобы лучше понять суть задачи, мы не будем сравнивать его с другими моделями.

Версия 2: более мощная, менее понятная

Создав эвристический алгоритм и изучив данные сайтов сети Stack Exchange, мы смогли определиться с первоначальным подходом к моделированию. Мы обучили простую модель, которую можно найти в соответствующем ноутбуке в GitHub-репозитории книги.

Эта модель использовала признаки, сгенерированные путем векторизации текста с помощью методов, описанных в разделе «Векторизация» на с. 95, в сочетании с созданными вручную признаками, которые были выделены в ходе исследования данных. При первом исследовании датасета я заметил следующие закономерности:

- Более длинные вопросы получают более высокие баллы.
- Вопросы о правильном использовании английского языка получают более низкие баллы.
- Вопросы, содержащие как минимум один вопросительный знак, получают более высокие баллы.

Я создал признаки, которые отражали эти предположения путем подсчета длины текста, проверки на наличие таких слов, как *punctuate* («ставить знаки препинания») и *abbreviate* («сократить»), и подсчета количества вопросительных знаков.

В дополнение к этим признакам я векторизовал входные вопросы, используя метод TF-IDF. Применение простейшей векторизации привязывает важность признаков к отдельным словам, что, в свою очередь, позволяет выдавать рекомендации на уровне слов с использованием описанных ранее методов.

Этот второй подход показал приемлемую совокупную производительность с точностью 0,62. В то же время его калибровочный график оставлял желать лучшего, в чем можно убедиться, взглянув на рис. 7.3.

Изучение важности признаков этой модели показало, что из созданных вручную признаков прогностически значимой была только длина вопроса. Все остальные признаки не повлияли на модель. Повторное изучение датасета позволило выявить еще несколько потенциально прогностически значимых признаков:

- Умеренное использование знаков препинания предположительно обеспечивает более высокий балл.
- Эмоционально окрашенные вопросы предположительно получают более низкий балл.
- Вопросы с большим количеством наречий и описаний предположительно получают более высокий балл.

Чтобы отразить эти новые гипотезы, я сформировал новый набор признаков. Я создал счетчики для каждого возможного знака пунктуации, а также счетчики количества используемых в предложении слов той или иной части речи (например, глаголов или прилагательных). Наконец, я добавил признак, отражающий эмоциональность вопроса. Подробное описание этих признаков можно найти в ноутбуке по второй модели в GitHub-репозитории книги.

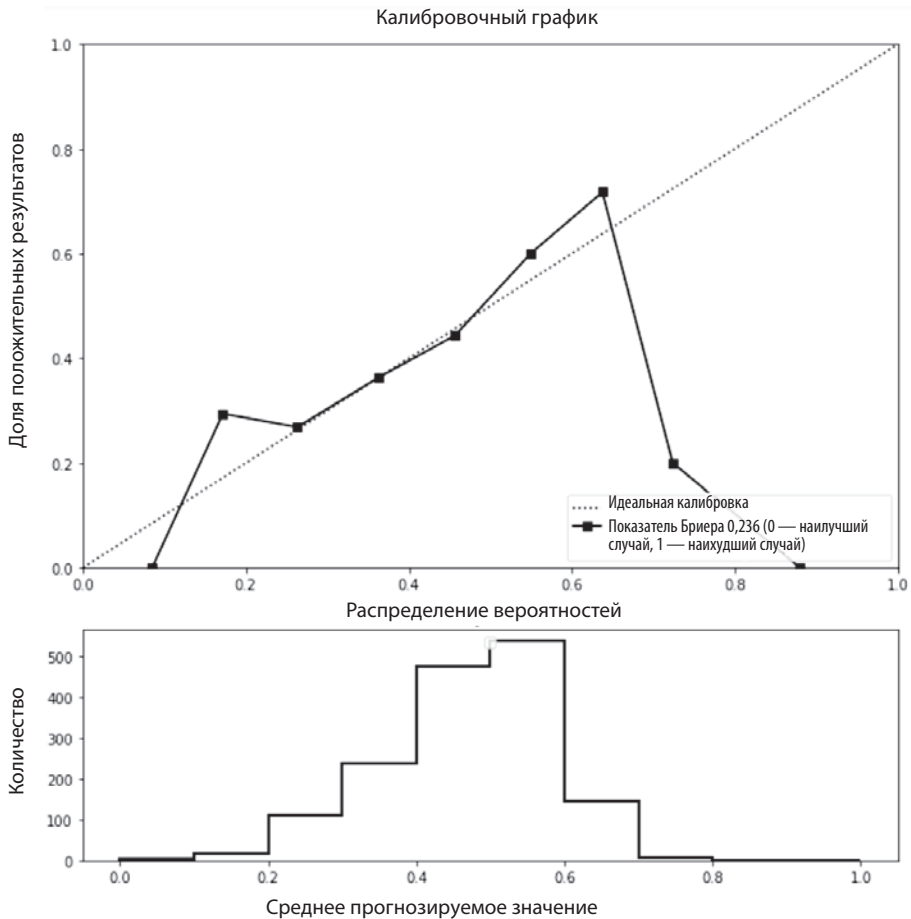


Рис. 7.3. Калибровочный график второй версии модели

Обновленная версия модели показала чуть более высокую совокупную производительность с точностью 0,63. Однако ее калибровочная кривая ничуть не лучше, чем калибровочная кривая предыдущей модели. Изучение важности признаков этой модели показало, что она опирается исключительно на созданные вручную признаки, а значит, эти признаки обладают некоторой прогностической значимостью.

Когда модель опирается на такие понятные признаки, нам проще сформулировать рекомендации для пользователя, чем в случае использования векторизованных признаков на уровне слов. Например, наиболее важными признаками на уровне слов для этой модели являются слова *are* («являются¹») и *what* («что»). Хотя мы можем догадываться, почему наличие этих слов коррелирует с качеством вопросов, совет чаще или реже употреблять в вопросе определенные слова будет не слишком понятной рекомендацией.

Желая избавиться от ограничений векторизованного представления и учитывая то, что созданные вручную признаки в данном случае обладают прогностической значимостью, я попытался создать более простую модель, не использующую какие-либо векторизованные признаки.

Версия 3: понятные рекомендации

Третья модель использует только созданные вручную признаки (то есть счетчики знаков пунктуации и частей речи, эмоциональную окрашенность и длину вопроса). И это всего 30 признаков, а не 7000, как в случае векторизованных представлений. Более подробное описание данной модели можно найти в соответствующем ноутбуке в GitHub-репозитории книги. Отказ от векторизованных признаков в пользу тех, которые созданы вручную, позволяет МО-редактору применять только понятные для пользователя признаки. В то же время это может несколько снизить производительность модели.

С точки зрения общей производительности данная модель работает хуже предыдущих, ее точность равна 0,597. Однако она намного лучше откалибрована. Из рис. 7.4 видно, что модель 3 хорошо откалибрована почти для всех вероятностей, включая значения, превышающие 0,7, с которыми другие модели справляются с трудом. Как показывает гистограмма, это объясняется тем, что данная модель чаще других предсказывает такие вероятности.

Благодаря широкому диапазону выдаваемых баллов и их улучшенной калибровке, она больше других моделей подходит для отображения пользователям балла в качестве подсказки. Такая модель будет наилучшим выбором также по критерию понятности рекомендаций, поскольку она использует только поддающиеся объяснению признаки. Наконец, данная модель работает быстрее других, поскольку использует меньше признаков.

Поскольку модель 3 лучше всего подходит для МО-редактора, именно ее следует использовать при развертывании первой версии продукта. В следующем разделе будет кратко описано, как применять эту модель в сочетании с методами выдачи рекомендаций по редактированию текста.

¹ С глагола «are» начинаются многие вопросы на английском языке. — *Примеч. ред.*

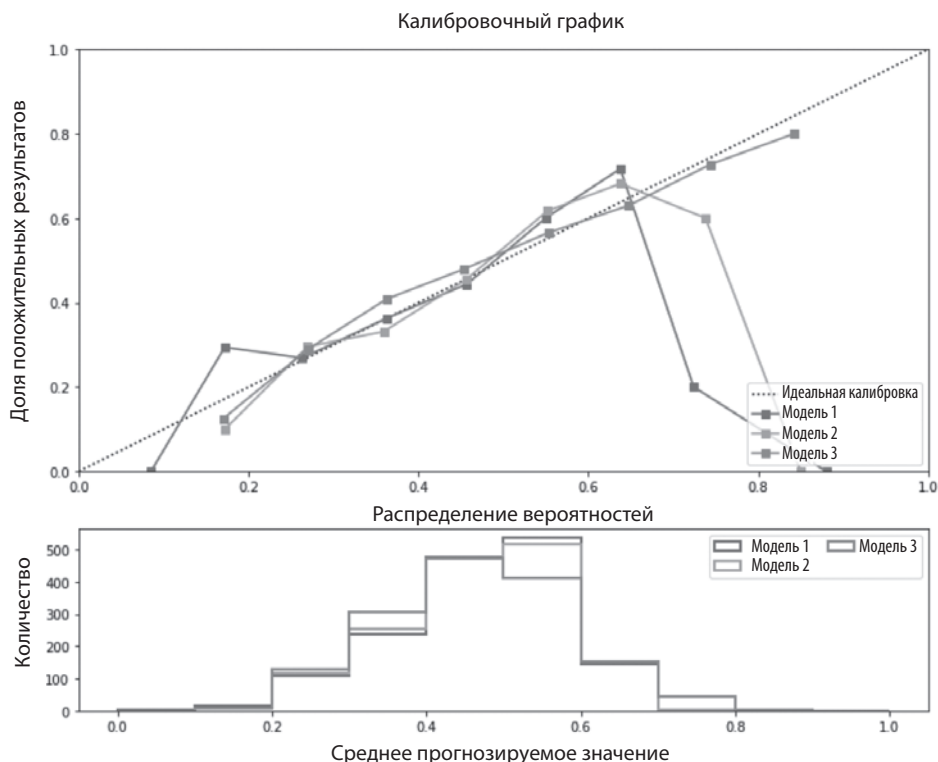


Рис. 7.4. Сравнение калибровочных кривых

Создание рекомендаций по редактированию текста

Для МО-редактора можно с успехом использовать любой из четырех описанных выше методов генерирования рекомендаций. Все эти методы также представлены в соответствующем ноутбуке в GitHub-репозитории книги. Поскольку выбранная нами модель работает быстро, мы можем разобрать здесь наиболее сложный подход, использующий интерпретаторы «черного ящика».

Для начала давайте рассмотрим общую функцию для выдачи рекомендаций, которая принимает вопрос и выдает рекомендации по редактированию текста на основе обученной модели. Эта функция выглядит следующим образом:

```
def get_recommendation_and_prediction_from_text(input_text, num_feats=10):
    global clf, explainer
    feats = get_features_from_input_text(input_text)
    pos_score = clf.predict_proba([feats])[0][1]
```

```
exp = explainer.explain_instance(
    feats, clf.predict_proba, num_features=num_feats, labels=(1,))
parsed_exps = parse_explanations(exp.as_list())
recs = get_recommendation_string_from_parsed_exps(parsed_exps)
return recs, pos_score
```

Если мы вызовем эту функцию на образце входных данных и просто выведем на экран ее результаты, то получим ряд рекомендаций, выглядящих примерно так, как показано ниже. После этого можно отобразить эти рекомендации пользователю, чтобы он мог доработать свой вопрос.

```
>> recos, score = get_recommendation_and_prediction_from_text(example_question)
>> print("%s score" % score)
0.4 score
>> print(*recos, sep="\n")
Increase question length
Increase vocabulary diversity
Increase frequency of question marks
No need to increase frequency of periods
Decrease question length
Decrease frequency of determiners
Increase frequency of commas
No need to decrease frequency of adverbs
Increase frequency of coordinating conjunctions
Increase frequency of subordinating conjunctions
```

Давайте разберемся с тем, что делает эта функция. Посмотрев на ее сигнатуру, можно заметить, что функция принимает в качестве параметров входную строку, представляющую вопрос, и необязательный аргумент, указывающий, для какого количества наиболее важных признаков следует создавать рекомендации. Она возвращает рекомендации, а также показатель, отражающий текущее качество вопроса.

Теперь давайте взглянем на тело функции. В первой строке объявляются две глобальные переменные, одна из которых представляет обученную модель, а вторая — такой же экземпляр интерпретатора LIME, какой мы определили в разделе «Извлечение локальной важности признаков» на с. 188. Следующие две строки генерируют признаки на основе входного текста и передают их классификатору, чтобы он выдал предсказания. Затем объявляется переменная `exp` с объяснениями, сгенерированными интерпретатором LIME.

Последние два вызова функций преобразуют эти объяснения в читаемые рекомендации. Чтобы выяснить, как именно это делается, давайте взглянем, как эти функции определяются. Вот как выглядит код функции `parse_explanations`:

```
def parse_explanations(exp_list):
    global FEATURE_DISPLAY_NAMES
    parsed_exps = []
```

```

for feat_bound, impact in exp_list:
    conditions = feat_bound.split(" ")

    # Мы игнорируем условия с двумя ограничениями, например 1 <= a <3,
    # потому что их труднее сформулировать в виде рекомендации
    if len(conditions) == 3:
        feat_name, order, threshold = conditions

        simple_order = simplify_order_sign(order)
        recommended_mod = get_recommended_modification(simple_order, impact)

        parsed_exps.append(
            {
                "feature": feat_name,
                "feature_display_name": FEATURE_DISPLAY_NAMES[feat_name],
                "order": simple_order,
                "threshold": threshold,
                "impact": impact,
                "recommendation": recommended_mod,
            }
        )
    return parsed_exps

```

Хотя функция длинная, она решает сравнительно простую задачу. Она принимает возвращаемый интерпретатором LIME массив степеней важности и выдает более структурированный словарь, который может быть использован в рекомендациях. Вот как выглядит пример такого преобразования:

```

# массив exps соответствует формату объяснений LIME
>> exps = [('num_chars <= 408.00', -0.03908691525058592),
            ('DET > 0.03', -0.014685507408497802)]

>> parse_explanations(exps)

[{'feature': 'num_chars',
  'feature_display_name': 'question length',
  'order': '<',
  'threshold': '408.00',
  'impact': -0.03908691525058592,
  'recommendation': 'Increase'},
 {'feature': 'DET',
  'feature_display_name': 'frequency of determiners',
  'order': '>',
  'threshold': '0.03',
  'impact': -0.014685507408497802,
  'recommendation': 'Decrease'}]

```

Обратите внимание, что данный вызов функции преобразовал выданное интерпретатором LIME пороговое значение в рекомендацию о необходимости увеличить или уменьшить значение признака. Это делается путем вызова функции `get_recommended_modification`. Определяется она так:

```
def get_recommended_modification(simple_order, impact):
    bigger_than_threshold = simple_order == ">"
    has_positive_impact = impact > 0

    if bigger_than_threshold and has_positive_impact:
        return "No need to decrease"
    if not bigger_than_threshold and not has_positive_impact:
        return "Increase"
    if bigger_than_threshold and not has_positive_impact:
        return "Decrease"
    if not bigger_than_threshold and has_positive_impact:
        return "No need to increase"
```

(No need to decrease — не надо уменьшать; Increase — увеличить; Decrease — уменьшить; No need to increase — не надо увеличивать).

После преобразования объяснений в рекомендации нам остается лишь отобразить рекомендации в подходящем формате. Это делается с помощью последнего вызова функции внутри функции `get_recommendation_and_prediction_from_text`. Вот как выглядит код этой функции:

```
def get_recommendation_string_from_parsed_exps(exp_list):
    recommendations = []
    for feature_exp in exp_list:
        recommendation = "%s %s" % (
            feature_exp["recommendation"],
            feature_exp["feature_display_name"],
        )
        recommendations.append(recommendation)
    return recommendations
```

При желании вы можете поэкспериментировать с редактором и доработать его — соответствующий код можно найти в ноутбуке по генерированию рекомендаций в GitHub-репозитории книги. В конце этого ноутбука приводится пример многократного использования рекомендаций модели для изменения формулировки вопроса с целью повысить его балл. Я приведу этот пример и здесь, чтобы наглядно продемонстрировать, как такие рекомендации могут использоваться, и помочь пользователям улучшить формулировку вопросов.

```
// Первая версия вопроса
>> get_recommendation_and_prediction_from_text(
    """
    I want to learn how models are made
    """
)

0.39 score
Increase question length
Increase vocabulary diversity
Increase frequency of question marks
```

No need to increase frequency of periods
 No need to decrease frequency of stop words

```
// После выполнения первых трех рекомендаций
>> get_recommendation_and_prediction_from_text(
    """
```

```
I'd like to learn about building machine learning products.
Are there any good product focused resources?
Would you be able to recommend educational books?
    """
)
```

```
0.48 score
Increase question length
Increase vocabulary diversity
Increase frequency of adverbs
No need to decrease frequency of question marks
Increase frequency of commas
```

```
// После повторного выполнения рекомендаций
>> get_recommendation_and_prediction_from_text(
    """
```

```
I'd like to learn more about ML, specifically how to build ML products.
When I attempt to build such products, I always face the same challenge:
how do you go beyond a model?
What are the best practices to use a model in a concrete application?
Are there any good product focused resources?
Would you be able to recommend educational books?
    """
)
```

```
0.53 score
```

Вот и всё! Теперь у нас есть пайплайн, который может принять вопрос и выдать пользователю действенные рекомендации. Конечно, этот пайплайн еще нельзя назвать совершенным, но тем не менее мы уже располагаем вполне работоспособной полнофункциональной версией редактора на базе МО. При желании вы можете попытаться самостоятельно улучшить эту текущую версию — попробуйте поработать с ней и найти, какие ошибки необходимо устранить. Интересно отметить, что, хотя любую модель можно доработать, на производительности данного редактора лучше всего скажется создание новых понятных для пользователя признаков.

Заклучение

В этой главе мы рассмотрели различные методы генерирования рекомендаций с помощью обученной классификационной модели. Принимая во внимание эти методы, мы сравнили подходы к моделированию и выбрали тот, который

наилучшим образом реализует цель нашего продукта, то есть помогает пользователям улучшить формулировку вопросов. Затем создали сквозной пайплайн для МО-редактора и применили его для выдачи рекомендаций.

Выбранная нами модель еще далека от совершенства и ее нужно дорабатывать. Если вы хотите попрактиковаться в использовании концепций, рассмотренных в части III, попробуйте выполнить эти циклы самостоятельно. Каждая из глав части III освещает определенную составляющую итерационного цикла МО. Для успешной реализации проекта МО необходимо многократно выполнять шаги, описанные в этой части книги, пока вы не увидите, что модель готова к развертыванию.

В части IV мы узнаем, какие риски влечет за собой развертывание модели и каким образом их можно устранять, а также как выявлять колебания производительности модели и принимать соответствующие меры.

ЧАСТЬ IV

Развертывание и мониторинг

После того как вы создали модель и убедились в ее работоспособности, необходимо сделать ее доступной для пользователей. Для этого существуют разные способы. Простейший подход — создать небольшой API, однако этого мало, чтобы гарантировать, что модель будет хорошо работать для всех пользователей.

На рис. IV.1 показаны некоторые системы, которые обычно дополняют модель при эксплуатации. Мы рассмотрим их в следующих главах книги.

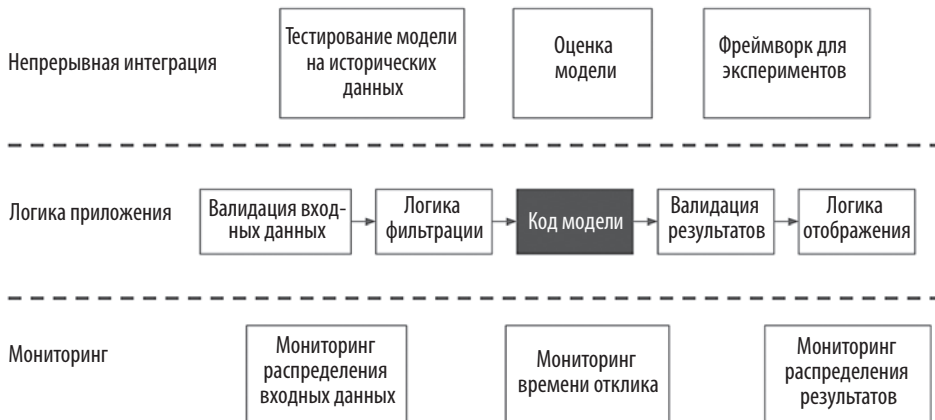


Рис. IV.1. Типичный пайплайн моделирования для выпуска в производство

Пайплайн МО в производственной среде должен уметь выявлять ошибки в данных и в модели и «элегантно» их обрабатывать. В идеале вы должны прогнозировать ошибки заранее и иметь некую стратегию развертывания обновленных

моделей. Если что-то из сказанного кажется вам пугающе сложной задачей, не волнуйтесь! Далее мы разберемся с тем, как это делается.

Глава 8

Перед развертыванием следует окончательно все проверить: оценить вероятность того, что модели будут использованы неправильно или не по назначению, и по возможности принять предупредительные меры.

Глава 9

Здесь мы рассмотрим методы и платформы для развертывания моделей и то, как следует подходить к их выбору.

Глава 10

В этой главе мы узнаем, как создать надежную производственную среду для поддержки модели, которая будет включать в себя поиск и устранение ошибок модели, оптимизацию ее производительности и систематическое проведение дообучения.

Глава 11

Заключительная глава будет посвящена такому важному процессу, как мониторинг. В частности, мы узнаем, зачем нужен мониторинг моделей, как это лучше делать и как связать нашу систему мониторинга со стратегией развертывания.

Что еще учесть при развертывании модели

В предыдущих главах мы разобрали, как гарантировать эффективность обучения модели и обобщения. Это необходимые, но недостаточные шаги для гарантии успешного МО-продукта.

При развертывании модели вы должны тщательно проанализировать, какие типы ошибок могут иметь последствия для пользователей. Вот некоторые из вопросов, на которые нужно ответить при создании продуктов, обучающихся на данных:

- Как были собраны используемые данные?
- Какие допущения делает модель, обучаясь на этом датасете?
- Является ли этот датасет достаточно репрезентативным для получения хорошей модели?
- Как результаты вашей работы могут быть использованы не по назначению?
- Каковы назначение и область применения вашей модели?

Найти ответы на часть таких вопросов стремится «этика данных». Методы этой дисциплины постоянно развиваются, и если вы хотите подробнее ознакомиться с ними, я могу порекомендовать книгу Майка Лукидеса (Mike Loukides) и др. «*Этика и наука о данных*» («*Ethics and Data Science*»¹) (O'Reilly).

В этой главе мы обсудим ряд важных моментов, касающихся сбора и использования данных, и узнаем, какие сложности таит в себе попытка проследить за тем, чтобы модели хорошо работали для всех пользователей. Завершается эта часть книги полезным интервью, где даются советы о том, как превратить предсказания модели в обратную связь от пользователей.

¹ <https://www.oreilly.com/library/view/ethics-and-data/9781492043898/>

Давайте начнем с вопросов, связанных с данными. Сначала мы поговорим о праве собственности на них, а затем — о смещении.

Забота о данных

Этот раздел мы начнем с рекомендаций по хранению, использованию и генерированию данных. Сначала поговорим о праве собственности на данные и о том, какие обязательства влечет за собой их хранение. Затем обсудим, что обычно ведет к появлению смещения в данных и как можно его учесть при моделировании. Наконец, узнаем, к каким негативным последствиям может привести смещение и почему так важно его устранить.

Право собственности на данные

Под «правом собственности» понимаются требования по сбору и использованию данных. Вот некоторые наиболее важные вопросы, которые следует рассмотреть:

- *Сбор информации.* Имеете ли вы право с юридической точки зрения на сбор и использование того датасета, на котором вы хотите обучить свою модель?
- *Разрешение на использование данных.* Объяснили ли вы своим пользователям, почему вам нужны их персональные данные и как вы собираетесь использовать эти данные? Дали ли пользователи свое согласие?
- *Хранение данных.* Как вы храните свои данные, кто имеет к ним доступ и когда вы их удалите?

Собирая данные своих пользователей, вы упрощаете для себя задачу персонализации и адаптации пользовательского опыта. В то же время это влечет за собой как моральные, так и юридические обязательства. Моральное обязательство обеспечить безопасное хранение данных существовало всегда, однако сегодня это все чаще становится и юридическим обязательством. Так, например, принятый в Европе Общий регламент по защите данных (GDPR) устанавливает ряд строгих правил в отношении сбора и обработки данных.

Организация, хранящая большие объемы данных, рискует столкнуться с серьезными проблемами в случае их утечки. Помимо подрыва доверия к организации, такие утечки часто ведут к предъявлению судебных исков. В силу этого ограничение объемов собираемых данных снижает риск правовых последствий.

Что касается нашего МО-редактора, мы начнем с использования общедоступных датасетов, которые были собраны и размещены в интернете с согласия пользователей. Если нам потребуется собрать какие-либо дополнительные данные, призванные помочь в улучшении датасета, например сведения об использовании

нашего сервиса, мы должны будем четко определить политику сбора данных и ознакомить пользователей с ее содержанием.

Наряду с вопросами сбора и хранения данных важно рассмотреть вопрос о том, может ли снижаться производительность модели при использовании собранных данных. Часто бывает так, что датасет хорошо подходит для одних случаев, но не подходит для других. Давайте разберемся, почему так происходит.

Смещение данных

Каждый датасет — результат определенных решений по сбору данных. Принятие этих решений ведет к тому, что получаемый датасет отражает реальность с некоторой предвзятостью, или, иначе говоря, смещением. МО-модель обучается на датасете и, соответственно, воспроизводит это смещение.

Допустим, модель обучается предсказывать наличие лидерских качеств на исторических данных, прогнозируя вероятность того, что человек станет руководителем организации. Предсказание строится на основе некоторой информации о человеке, включая его пол. Как показывает доклад «Данные о женщинах-лидерах»¹, составленный Исследовательским центром Пью (Pew Research Center), исторически сложилось так, что руководителями большинства крупнейших компаний США из списка Fortune 500 являются мужчины. Обучение модели на этих данных приведет к тому, что принадлежность к мужскому полу будет расцениваться как важный предиктор наличия лидерских качеств. Принадлежность к мужскому полу и должность топ-менеджера в этом датасете коррелируют в силу социально обусловленных причин. Это приведет к снижению вероятности назначения женщины на руководящий пост. Если мы просто обучим модель на этих данных и затем будем использовать ее для выдачи предсказаний, то тем самым мы лишь усилим существующую необъективность.

Как бы ни был велик соблазн принять используемые данные за эталон, в действительности большинство датасетов представляют собой набор приблизительных оценок, полученных без учета общего контекста. Поэтому, предполагая, что любой датасет имеет смещение, мы должны начать с оценки влияния этого смещения на модель. Затем можно попытаться улучшить датасет за счет повышения его репрезентативности и скорректировать модель так, чтобы она была менее склонна воспроизводить имеющееся смещение.

Вот некоторые из распространенных причин появления ошибок и смещения в датасетах:

- *Ошибки измерения или искаженные данные.* Каждый элемент данных обладает некоторой погрешностью, обусловленной способом его получения.

¹ <https://oreil.ly/vTLkH>

Большинство моделей игнорирует эту погрешность, из-за чего она может стать систематической.

- *Репрезентативность.* Большинство датасетов содержит данные только по некоторой части населения. Так, когда для систем распознавания лиц изначально использовались датасеты с изображениями лиц преимущественно представителей белой расы, модели хорошо работали для этой группы населения и плохо для других.
- *Доступ.* Одни датасеты получить труднее, чем другие. Например, в интернете проще собрать тексты на английском языке, чем на других языках. В силу этого наиболее современные языковые модели, как правило, обучают исключительно на англоязычных данных. Как следствие, англоговорящие пользователи получают в свое распоряжение более совершенные сервисы на базе МО по сравнению с остальными пользователями. Этот разрыв еще больше усиливается за счет того, что дополнительный объем пользователей англоязычной версии продукта делает эту модель еще более совершенной по сравнению с версиями для других языков.

Поскольку производительность моделей оценивается на тестовом наборе, необходимо особенно тщательно позаботиться о том, чтобы он был как можно более точным и репрезентативным.

Тестовые наборы

Вопрос репрезентативности актуален для любой задачи МО. В разделе «Разделение датасета» на с. 128 было рассказано, почему для оценки производительности модели следует разделить данные на несколько наборов. Важно, чтобы после разделения тестовый набор содержал достаточно обширные, репрезентативные и реалистичные данные, поскольку они будут использоваться для оценки производительности в реальных условиях.

Для этого при проектировании тестовой выборки нужно принять во внимание всех потенциальных пользователей модели. Чтобы повысить шансы на то, что опыт каждого пользователя будет положительным, постарайтесь включить в тестовый набор образцы, представляющие все типы пользователей.

При проектировании тестового набора также следует учесть цели продукта. Так, при создании модели для постановки диагноза вам нужно убедиться, что она эффективна для пациентов обоих полов. Для этого в тестовом наборе должно быть достаточно данных о пациентах обоих полов. Очень полезен сбор мнений пользователей. Прежде чем развертывать модель, постарайтесь дать широкому кругу пользователей возможность ознакомиться и поработать с ней, а затем поделиться своими впечатлениями.

И еще одно, последнее замечание касательно смещения. Модели часто обучают на исторических данных, отражающих положение дел в прошлом. В силу этого

смещение обычно затрагивает те группы населения, которые и без того ущемлены в своих правах. Поэтому устранение смещения может помочь сделать системы более справедливыми по отношению к людям, которые больше всего в этом нуждаются.

Систематическое смещение

Под систематическим смещением понимается ситуация, когда политика государства или корпораций ведет к дискриминации определенных групп населения. В результате этой дискриминации исторических данных о таких группах населения часто не хватает. Так, например, если в силу социальных факторов в базе исторических данных по уголовным преступлениям будут избыточно представлены определенные группы населения, то обучение модели на этих данных приведет к тому, что она закодирует это смещение и будет воспроизводить его при прогнозировании будущих преступлений.

Это может иметь катастрофические последствия и привести к маргинализации таких групп населения. В статье «Предвзятость машин» («Machine Bias») Дж. Энгвин (J. Angwin) и др., размещенной на сайте ProPublica¹, описан конкретный пример воспроизведения такого смещения при прогнозировании преступлений.

Устранение или уменьшение смещения в датасете — непростая задача. Некоторые разработчики пытаются не допустить предвзятости отношения модели к таким признакам, как этническая и половая принадлежность, путем исключения их из числа признаков, используемых моделью для выдачи предсказаний.

Однако, как показывает практика, простое исключение признака не позволяет гарантировать непредвзятое отношение модели, поскольку подобный признак обычно сильно коррелирует с рядом других признаков датасета. Так, например, в США этническая принадлежность сильно коррелирует с почтовым индексом и уровнем дохода. Если вы удалите только признак этнической принадлежности, модель будет по-прежнему проявлять предвзятость, просто ее будет труднее обнаружить.

Вместо этого вы должны четко определить, что в вашем случае служит препятствием к беспристрастности. Например, можно воспользоваться подходом, описанным в статье М. Б. Зафара (M. B. Zafar) и др. «Проблема беспристрастности: механизмы беспристрастной классификации» («Fairness Constraints: Mechanisms for Fair Classification»²), при котором степень беспристрастности модели оценивается с помощью так называемого правила $p\%$. Правило $p\%$ гла-

¹ <https://oreil.ly/6UE3z>

² <https://oreil.ly/JWlIi>

сит: «Соотношение между долей объектов, имеющих чувствительные атрибуты и получающих положительный результат, и долей объектов, не имеющих этого атрибута и получающих тот же результат, должно быть не меньше чем $p:100$ ». Это правило позволяет нам количественно оценить смещение и, соответственно, лучше справиться с его устранением. Однако необходимо четко понимать, по какому признаку не должно быть смещения.

Наряду с оценкой рисков, смещения и ошибок данных МО требует и оценки моделей.

Забота о модели

Как можно свести к минимуму вероятность нежелательного смещения?

Существуют различные виды негативного влияния модели на пользователей. Сначала мы рассмотрим случай попадания в цикл обратной связи, после чего изучим риски совершения моделью неявных ошибок на небольшом сегменте населения. Затем поговорим о том, почему важно обеспечить достоверность предсказаний МО-модели. Завершит этот раздел обсуждение риска вредоносного вмешательства злоумышленников в модель.

Циклы обратной связи

В большинстве систем на базе МО следование пользователей рекомендациям модели повышает вероятность того, что последующие версии модели будут выдавать такую же рекомендацию. Если не контролировать этот процесс, он может привести к тому, что модель попадет в самоусиливающийся цикл обратной связи.

Например, если первая версия модели рекомендаций по просмотру видеороликов будет немного чаще рекомендовать видео с кошками, чем с собаками, то пользователи в целом будут смотреть больше видеороликов с кошками, чем с собаками. Если затем мы обучим вторую версию модели, используя исторические данные по выданным рекомендациям и кликам пользователей, мы тем самым внесем смещение первой модели в свой датасет, в результате чего вторая модель будет еще более склонна отдавать предпочтение кошкам.

Поскольку модели для рекомендации контента часто обновляются по нескольку раз в день, уже довольно скоро очередная версия модели будет рекомендовать исключительно видеоролики с кошками. На рис. 8.1 показана схема этого цикла обратной связи. В силу того, что видеоролики с кошками изначально пользуются большей популярностью, модель постепенно учится рекомендовать их все чаще и в итоге, как показано справа, перестает рекомендовать какие-либо другие видеоролики.

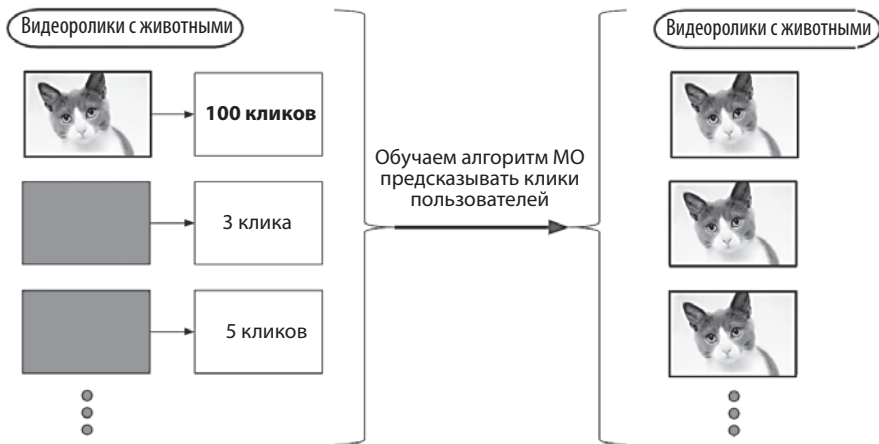


Рис. 8.1. Пример цикла обратной связи

Хотя переполненность интернета видеороликами с кошками — не такая уж трагедия, эти механизмы могут быстро увеличить негативное смещение и начать рекомендовать к просмотру неприемлемый или опасный контент. Стремясь обеспечить максимальную кликабельность, модель обучается рекомендовать кликбейтный контент с заманчивыми заголовками, но бесполезный для пользователя.

Циклы обратной связи также имеют тенденцию к смещению в сторону меньшинства наиболее активных пользователей. Если рекомендательный алгоритм платформы для просмотра видеороликов будет обучаться на основе того, сколько раз пользователи кликнули по каждому видеоролику, то в итоге рекомендации могут чрезмерно подстраиваться под группу самых активных пользователей, которые совершают большую часть кликов. Эти же рекомендации будут предлагаться и всем другим пользователям платформы, независимо от их индивидуальных предпочтений.

Для снижения негативного влияния циклов обратной связи старайтесь выбирать метки, сводящие к минимуму вероятность попадания в такой цикл. Количество кликов пользователей говорит о том, сколько раз был просмотрен тот или иной видеоролик, но это отнюдь не означает, что он каждому понравился. Выбор количества кликов в качестве целевого параметра оптимизации приведет к тому, что модель будет рекомендовать визуально наиболее броский контент, не обращая внимания на степень его релевантности. Нивелировать влияние этого цикла обратной связи можно, выбрав в качестве целевой метрики не количество кликов, а длительность просмотра, которая больше коррелирует со степенью удовлетворенности пользователей.

Однако даже когда рекомендательный алгоритм оптимизируется на основе некоего показателя вовлеченности пользователей, не исключена вероятность

возникновения цикла обратной связи, поскольку единственной целью при этом является максимизация метрики, которая может быть бесконечно большой. Так, например, даже когда алгоритм оптимизируется на основе длительности просмотра, чтобы стимулировать просмотр более интересного контента, эта метрика будет максимальной в том случае, когда каждый пользователь будет тратить на просмотр видеороликов все свое время. Хотя применение таких метрик вовлеченности пользователей часто может быть выгодным, возникает вопрос о том, всегда ли они будут подходящей целью оптимизации.

Наряду с риском возникновения цикла обратной связи также существует риск того, что производительность модели в реальных условиях будет ниже, чем ожидалось, несмотря на получение хороших результатов согласно валидационным офлайн-метрикам.

Инклюзивная производительность модели

В разделе «Оценка модели: не ограничивайтесь точностью» на с. 140 мы рассмотрели различные метрики, позволяющие оценить производительность на различных подмножествах датасета. Такой анализ помогает убедиться в том, что модель работает одинаково хорошо для различных типов пользователей.

Это особенно важно при оценке развертываемости новых версий существующих моделей. Если сравнивать модели лишь на основе их общей производительности, можно не заметить существенного ухудшения качества модели на определенном сегменте данных.

Упущение из виду такого снижения производительности не раз становилось причиной катастрофического сбоя продукта. Так, в 2015 году система автоматического тегирования изображений классифицировала фотографии пользователей-афроамериканцев как горилл (см. статью от 2015 года на сайте BBC¹). Эта ужасная ошибка произошла вследствие того, что модель не была проверена на репрезентативном наборе входных данных.

Проблемы такого рода могут возникать и при обновлении существующей модели. Допустим, вы обновляете модель для распознавания лиц. У предыдущей модели доля верных результатов составляла 90%, а у новой модели этот показатель составляет 92%. Прежде чем развертывать эту новую модель, вы должны сравнить ее производительность на разных подмножествах пользователей. При этом может выясниться, что, несмотря на небольшое улучшение общей производительности, новая модель демонстрирует очень плохие результаты на фотографиях женщин, возраст которых превышает 40 лет. От развертывания такой модели придется воздержаться. Вместо этого

¹ <https://oreil.ly/nVkZv>

вы должны модифицировать обучающие данные, сделав их более репрезентативными, и дообучить модель так, чтобы она хорошо работала для всех категорий пользователей.

Пренебрежение таким сравнительным анализом может привести к тому, что модель не будет полезна для значительной части целевой аудитории. Хотя ни одна модель не способна работать буквально для всех возможных входных данных, важно убедиться в том, что ваша модель справляется со всеми ожидаемыми видами данных.

О контексте

Пользователи не всегда будут знать, что тот или иной фрагмент данных был получен как предсказание МО-модели. По возможности следует информировать пользователя о том, в каком контексте было получено предсказание, и тогда он сможет принять обоснованное решение о том, как его использовать. Для начала можно рассказать пользователю, как обучалась модель.

Пока еще не существует общепринятого формата дисклеймера для модели, однако, согласно проводимым исследованиям в этой области, существуют перспективные форматы, такие как карты моделей. Этот формат представляет собой систему документирования для предоставления прозрачной отчетности по моделям (см. статью М. Митчелл (M. Mitchell) и др. «Карты моделей для отчетности по моделям» («Model Cards for Model Reporting»¹)). Данная методика предлагает сопровождать модель метаданными о том, как она была обучена, на каких данных она тестировалась, для чего она предназначена и т. д.

В случае нашего учебного примера МО-редактор предоставляет обратную связь на основе конкретного набора вопросов. Если бы нам нужно было развернуть его в качестве программного продукта, мы могли бы добавить дисклеймер про типы входных данных, на которых наша модель должна показывать хорошие результаты. Например, использовать такую простую формулировку: «Данный продукт предназначен для выдачи рекомендаций по улучшению формулировки вопросов. Он был обучен на вопросах, взятых с сайта сети Stack Exchange, посвященного литературному творчеству, и в силу этого может отражать конкретные предпочтения этого сообщества».

Итак, не забывайте о том, насколько важно информировать честных пользователей. А теперь давайте посмотрим, какие трудности в теории могут создавать менее дружелюбные пользователи.

¹ <https://arxiv.org/abs/1810.03993>

Мошенники

При реализации некоторых проектов МО следует принимать во внимание риск мошенничества. Так, злоумышленники могут попытаться взломать модель, предназначенную для выявления подозрительных транзакций по кредитным картам. Они также могут попытаться использовать обученную модель для незаконного доступа к исходным обучающим данным, к примеру конфиденциальной информации о пользователях.

Взлом модели

Многие из практически используемых МО-моделей служат для защиты от мошенничества банковских счетов и транзакций. А мошенники, в свою очередь, предпринимают попытки обманном путем заставить такие модели принять их за законных пользователей.

Если, к примеру, вам нужно предотвратить незаконный вход на онлайн-платформу, то, вероятно, вам следует принять в расчет некоторые дополнительные признаки, например страну проживания пользователя (при проведении крупномасштабных атак злоумышленники часто используют несколько серверов, расположенных в одном регионе). В то же время, обучая модель с использованием таких признаков, вы рискуете внести предвзятость к честным пользователям, проживающим в тех же странах, где живут мошенники. Кроме того, если вы будете полагаться только на один такой признак, злоумышленники смогут легко обманывать вашу систему путем предоставления неверных сведений о своем местоположении.

Чтобы защититься от мошенников, важно регулярно обновлять модели. По мере того как злоумышленники будут изменять поведение, подстраиваясь под используемые вами защитные механизмы, обновляйте свои модели, чтобы они могли быстро классифицировать новое поведение как мошенническое. Для этого необходимы системы мониторинга, которые будут замечать изменения в паттернах поведения. Мы подробно обсудим это в главе 11. Во многих случаях для обеспечения защиты от злоумышленников требуется создавать новые признаки. Если вы забыли, как это делается, вы можете освежить свои знания о создании признаков в разделе «Используйте данные для принятия решений о признаках и моделях» на с. 113.

Хотя чаще всего модели атакуют, чтобы заставить их выдать неверное предсказание, существуют и другие мотивы. Так, цель некоторых видов атак — получить доступ к данным, на которых была обучена модель.

Злонамеренное использование модели

Мошенники могут не только взломать модель, но и использовать ее для получения конфиденциальной информации. Поскольку модель отражает те данные,

на которых она была обучена, с помощью ее предсказаний можно выявить закономерности исходного датасета. Чтобы проиллюстрировать эту идею, давайте представим, что у нас есть классификатор, обученный на датасете из двух образцов. Эти образцы относятся к разным классам и отличаются друг от друга лишь значением одного признака. Если вы предоставите злоумышленникам доступ к этой модели и предсказаниям, выдаваемым ею для произвольных входных данных, в конечном итоге они смогут сделать логический вывод о том, что этот признак является единственным прогностически значимым параметром используемого датасета. Сходным образом мошенники могут получить и данные о том, как распределены признаки в обучающих данных. Характер этого распределения часто является конфиденциальной информацией.

Допустим, чтобы обнаружить попытку незаконной авторизации, на сайте было добавлено обязательное поле для почтового индекса. Злоумышленник может предпринять ряд попыток авторизации с использованием различных учетных записей, чтобы посмотреть, какие почтовые индексы позволяют успешно авторизоваться. Так он сможет оценить распределение почтовых индексов в обучающем наборе данных и, соответственно, географическое распределение пользователей сайта.

Простейший способ снижения эффективности таких атак сводится к тому, чтобы наложить ограничение на количество запросов, выполняемых отдельными пользователями, и тем самым помешать им изучить значения признаков. Данный способ нельзя назвать очень надежным, поскольку опытные мошенники могут обойти это ограничение путем создания множества учетных записей.

Это далеко не все виды мошенничества. Если вы планируете предоставить результаты своей работы широкому кругу пользователей, также следует убедиться в том, что никто не сможет использовать их для совершения опасных действий.

Риск злоупотребления и двойного назначения

Термин «двойное применение» подразумевает, что технологии, которые были созданы для одной цели, могут использоваться для другой. Часто такая проблема возникает из-за того, что МО-модели способны хорошо работать на сходных датасетах (см. рис. 2.3).

Если вы создадите модель, позволяющую пользователю изменять звук своего голоса, делая его похожим на голоса своих друзей, не сможет ли он злоупотреблять такой возможностью, выдавая себя за других без их согласия? Если вы решите создать подобную модель, надо снабдить ее хорошим руководством и ресурсами, чтобы пользователи четко понимали, как правильно использовать модель.

Сходным образом при использовании любой модели, способной точно классифицировать лица людей, возникает вероятность слежки за гражданами. Хотя

изначально такая модель может быть создана в качестве составной части умного домофона, впоследствии она может быть использована для автоматической слежки в рамках общегородской сети камер. Модели, обученные на одном датасете, могут представлять опасность при повторном обучении на другом аналогичном наборе данных.

Пока не существует четких рекомендаций решения проблемы двойного применения. Если у вас есть основания полагать, что результаты вашей работы могут быть использованы для неэтичных целей, я призываю вас подумать о том, как усложнить их воспроизведение в случае использования модели не по назначению, или обсудить эту проблему в рамках сообщества. Недавно компания OpenAI приняла решение не делать общедоступной свою самую мощную языковую модель, поскольку это могло бы существенно упростить распространение дезинформации в интернете (см. статью «Более совершенные языковые модели и связанные с ними проблемы» («Better Language Models and Their Implications») на сайте компании OpenAI¹). Это было неординарное решение, однако я совсем не удивлюсь, если в будущем такие проблемы будут возникать все чаще и чаще.

В завершение главы почитайте интервью с техническим директором компании Textio Крисом Харландом (Chris Harland), который обладает богатым опытом в развертывании моделей для пользователей и представлении результатов с информативным контекстом.

Крис Харланд: опыт поставки продуктов

Крис имеет степень PhD в области физики и успел поучаствовать в реализации множества задач МО, включая создание систем машинного зрения для извлечения структурированной информации из квитанций с целью автоматизированного учета расходов. Ему довелось поработать в составе исследовательского подразделения компании Microsoft, где он осознал важную роль машинного обучения. Позднее Крис перешел в компанию Textio, которая специализируется на создании продуктов для дополнения текстов, помогающих пользователям писать более красочные резюме. Я встретился и поговорил с Крисом о том, какие рекомендации он может дать, исходя из своего опыта поставки продуктов на базе МО, и какие дополнительные способы проверки результатов он использует, помимо метрик точности.

Вопрос. Textio использует МО для выдачи пользователям прямых указаний. Чем это отличается от других задач МО?

Ответ. Когда вас интересуют только сами предсказания, которые, скажем, отвечают на вопрос, когда лучше покупать золото или на кого подписаться в Твиттере, вполне допустима некоторая дисперсия. Однако когда вы выдаете

¹ <https://openai.ly/W1Y6f>

рекомендации по редактированию текста, дело обстоит иначе, поскольку ваши рекомендации имеют широкий подтекст.

Выдав пользователю рекомендацию написать еще 200 слов, модель должна вести себя последовательно, позволяя ему реализовать этот совет. Она не может «передумать», выдав пользователю рекомендацию снизить количество слов, после того как он напишет еще 150.

Ваши указания также должны быть предельно ясными. Так, совет «сократить количество стоп-слов на 50%» будет не слишком понятным указанием, а вот совет «уменьшить длину этих 3 предложений» окажет пользователям более действенную помощь. Поэтому важно использовать понятные для человека признаки, сохраняя при этом высокую производительность.

Фактически помощник по написанию текстов на базе МО направляет пользователя в пространстве признаков от исходной точки к более выгодному положению с точки зрения модели. Иногда при этом приходится пройти через менее выгодные точки, что может негативно отражаться на пользовательском опыте. Вы должны учитывать эти ограничения при создании продукта.

Вопрос. Как лучше выдавать такие указания?

Ответ. При выдаче указаний метрика точности (precision) важнее, чем полнота (recall). В случае выдачи рекомендаций высокий показатель полноты подразумевает способность выдавать рекомендации по всем потенциально релевантным предметным областям, а также некоторым нерелевантным (которых может быть много). А высокая точность подразумевает выдачу рекомендаций в перспективных предметных областях, при этом нерелевантные области игнорируются.

В силу высокой цены ошибки очень важно, чтобы выдаваемые рекомендации были как можно точнее. Важность метрики точности обусловлена еще и тем, что по мере получения рекомендаций пользователи будут их запоминать и уже без какой-либо подсказки выполнять рекомендуемые действия над входными данными в дальнейшем.

Кроме того, при выявлении различных факторов мы оцениваем, несут ли они какую-либо реальную пользу. Если нет, мы выясняем, почему так происходит. В качестве примера можно привести нашу рекомендацию касательно соотношения между количеством распространенных и редких слов. Заметив, что пользователи слишком редко выполняют эту рекомендацию, мы выяснили, что причиной является ее недостаточная практическая применимость. Соответственно, мы усовершенствовали ее путем непосредственного выделения тех слов, которые рекомендовалось изменить.

Вопрос. Как вы находите новые способы выдачи указаний пользователям или новые признаки?

Ответ. Здесь полезен и нисходящий и восходящий подход.

Нисходящее изучение гипотез опирается на знания о предметной области и в основном сводится к подбору признаков на основе предыдущего опыта. Свои идеи при этом могут предлагать как разработчики продукта, так и маркетологи. Такая нисходящая формулировка гипотезы может выглядеть следующим образом: «Мы считаем, что в рекрутинговых письмах есть что-то такое, что способствует привлечению сотрудников». Поэтому при нисходящем подходе обычно важно найти практический способ извлечения такого признака. Только после этого мы можем оценить, насколько признак прогностически значим.

Восходящий подход сводится к изучению классификационного пайплайна с целью понять, какие признаки являются для него прогностически значимыми. Какие признаки являются наиболее значимыми для классификатора хорошего и плохого текста в случае, когда общее представление текста включает в себя и векторы слов, и лексемы, и обозначения частей речи? Эксперты в предметной области гораздо лучше других справятся с выявлением таких закономерностей на основе предсказаний модели. Задача состоит в том, чтобы каким-то образом сделать эти признаки понятными для человека.

Вопрос. Как вы определяете, что модель уже достаточно хорошая?

Ответ. Не стоит недооценивать то, насколько далеко вы можете продвинуться, используя небольшой набор текстов на нужном вам языке. Как выяснилось, во многих случаях хватает всего тысячи документов, относящихся к рассматриваемой вами предметной области. При этом очень полезно разметить этот небольшой датасет. После этого можно будет начать тестирование модели на данных, не входящих в выборку.

Постарайтесь максимально упростить проведение экспериментов. Учитывая, что подавляющее большинство идей по изменению продукта в итоге дают нулевой результат, не стоит слишком беспокоиться по поводу добавления новых функций.

Наконец, нет ничего плохого в том, чтобы создать плохую модель. С нее вы и должны начать. Исправление плохо работающих моделей позволит вам сделать свой продукт более устойчивым к проблемам и поможет быстрее усовершенствовать его.

Вопрос. Как вы отслеживаете состояние модели в эксплуатационном окружении?

Ответ. В реальных условиях следует предоставлять пользователям четкие предсказания и давать возможность переопределять их. Протоколируйте значения признаков, предсказания и переопределенные значения, для того чтобы можно было их контролировать и впоследствии анализировать. Если ваша модель выдает некоторый балл, то в качестве дополнительного признака можно каким-либо образом соотнести этот балл с частотой использования ваших рекомендаций. Так, например, если вы пытаетесь предсказать, откроет ли пользователь то или

иное электронное письмо, будет очень полезно получить от пользователей реальные данные об открываемых письмах, для того чтобы вы могли улучшить свою модель.

Окончательным критерием успеха продукта является его востребованность, однако этот показатель зависит от множества различных факторов и может быть получен лишь по прошествии значительного времени.

Заклучение

Эту главу мы начали с рассмотрения вопросов, касающихся использования и хранения данных. Затем мы ознакомились с причинами возникновения смещения в данных и рекомендациями по его выявлению и уменьшению. После этого рассмотрели проблемы, возникающие при использовании моделей в реальных условиях, и способы снижения рисков, связанных с предоставлением пользователям доступа к моделям. Наконец, мы узнали, как можно повысить устойчивость систем к ошибкам при проектировании их архитектуры.

Решение этих проблем дается непросто, и специалистам в сфере МО предстоит еще немало сделать для того, чтобы предотвратить все возможные формы злоупотребления. Прежде всего, все практикующие специалисты должны осознать наличие этих проблем и принять их во внимание при разработке своих продуктов.

Теперь мы готовы приступить к разворачиванию моделей. Для начала в главе 9 мы рассмотрим плюсы и минусы различных вариантов разворачивания. После этого в главе 10 поговорим о методах устранения некоторых рисков, связанных с разворачиванием моделей.

ГЛАВА 9

Выбор варианта развертывания

В предыдущих главах мы рассмотрели процесс, начиная с общей идеи продукта на базе МО до его конечной реализации, а также методы итеративной доработки МО-приложения до тех пор, пока оно не будет готово к развертыванию.

В этой главе мы поговорим о плюсах и минусах различных вариантов развертывания. Разные методы развертывания подходят для разных требований. Выбирая метод, вам потребуется учесть множество факторов: задержку отклика, требования к аппаратному обеспечению и сети, конфиденциальность, стоимость и сложность.

Цель развертывания модели — предоставление пользователям возможности взаимодействовать с ней. Мы рассмотрим общепринятые методы достижения этой цели, а также дадим советы по выбору подходящего метода развертывания.

Начнем с простейшего способа — создадим веб-сервер для выдачи прогнозов.

Развертывание на сервере

Развертывание на стороне сервера подразумевает разработку веб-сервера, способного принимать запросы от клиентов, прогонять их через пайплайн инференса и возвращать результаты. Это решение хорошо вписывается в парадигму веб-разработки, поскольку здесь модель — просто еще один эндпоинт приложения. Пользователи отправляют свои запросы на этот эндпоинт, рассчитывая получить в ответ результаты.

При развертывании моделей на стороне сервера обычно используется одна из двух возможных схем обработки — потоковая или пакетная. При использовании потоковой обработки запросы принимаются по мере их поступления и сразу же обрабатываются. Пакетная схема обработки используется гораздо реже и сводится к единовременной обработке большого количества запросов. Давайте начнем с рассмотрения потокового подхода.

Потоковое приложение или API

При потоковом подходе модель рассматривается как эндпоинт, на который пользователи могут отправлять свои запросы. При этом в качестве пользователей могут выступать как конечные пользователи приложения, так и некоторый внутренний сервис, который использует предсказания модели. Например, модель, прогнозирующая посещаемость веб-сайта, может использоваться внутренним сервисом, выбирающим количество серверов в зависимости от прогноза количества посетителей.

В потоковом приложении запрос проходит через ту же последовательность шагов, что и в случае пайплайна инференса, рассмотренного нами в разделе «Начинайте с простого пайплайна» на с. 61. Давайте еще раз вспомним, как выглядит эта последовательность шагов.

1. Валидация запроса. Проверить все переданные значения параметров и, возможно, убедиться в наличии у пользователя необходимых разрешений для запуска модели.
2. Сбор дополнительных данных. Запросить из других источников необходимые дополнительные данные, например информацию о пользователе.
3. Предобработка данных.
4. Запуск модели.
5. Постобработка результата. Убедиться в том, что результат находится в допустимых пределах, а также дополнить его контекстом, чтобы он был более понятен пользователю, например указать степень уверенности модели в результате.
6. Возвращение результата.

Схема этой последовательности обработки показана на рис. 9.1.



Рис. 9.1. Схема работы потокового API

Метод с использованием эндпоинтов прост в реализации, но нужна инфраструктура, способная обеспечить линейное масштабирование по мере роста количества пользователей, поскольку для каждого пользователя осуществляется отдельный вызов инференса. Если количество посетителей превысит мощность сервера по обработке запросов, то обработка будет выполняться с задержкой или даже давать сбой. Соответственно, для адаптации такого пайплайна к изменениям посещаемости нужна возможность легкого подключения и отключения дополнительных серверов, что требует некоего уровня автоматизации.

Однако в случае простой демоверсии МО-редактора, у которой лишь несколько одновременно подключившихся пользователей, потоковый подход вполне подойдет. Для развертывания своего МО-редактора мы воспользуемся Flask¹. Это легковесный фреймворк для создания веб-приложений на языке Python, который позволяет легко настроить API для модели, используя всего несколько строк кода.

Я проведу здесь лишь общий обзор этого кода для развертывания прототипа; в полном виде он приводится в GitHub-репозитории книги². Flask-приложение состоит из двух составных частей. Первым компонентом является API, который принимает запросы и отправляет их модели для обработки. Вторым компонентом является простой веб-сайт на базе HTML-кода, который обеспечивает ввод текста пользователями и отображение результатов. Чтобы определить такой API, не требуется много кода. Вот, например, как выглядят две функции, выполняющие основную часть работы по обслуживанию третьей версии МО-редактора:

```
from flask import Flask, render_template, request

@app.route("/v3", methods=["POST", "GET"])
def v3():
    return handle_text_request(request, "v3.html")

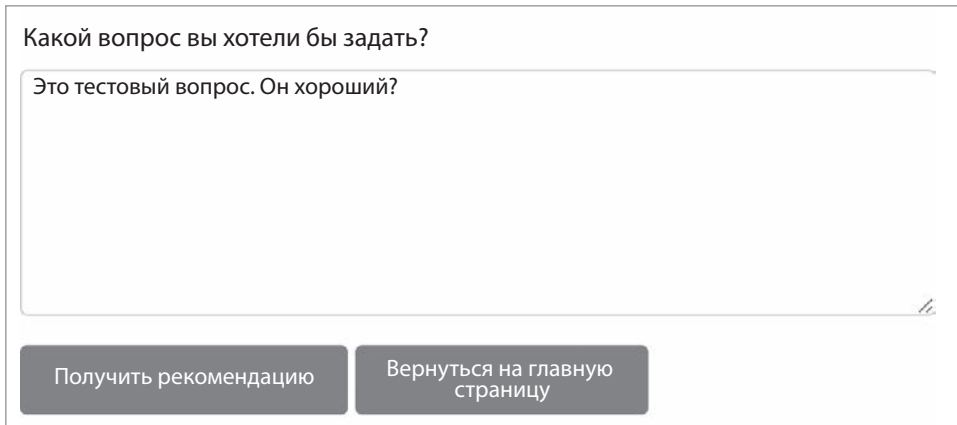
def handle_text_request(request, template_name):
    if request.method == "POST":
        question = request.form.get("question")
        suggestions = get_recommendations_from_input(question)
        payload = {"input": question, "suggestions": suggestions}
        return render_template("results.html", ml_result=payload)
    else:
        return render_template(template_name)
```

Функция `v3` определяет путь до HTML-кода, который должен отображаться при посещении пользователем страницы с адресом `/v3`. Решение о том, что

¹ <https://oreil.ly/cKLMn>

² <https://github.com/hundredblocks/ml-powered-applications>

отображать, принимается с помощью функции `handle_text_request`. При первом посещении пользователем этой страницы используется запрос типа GET, и, соответственно, функция отображает HTML-шаблон. Скриншот этой HTML-страницы показан на рис. 9.2. Когда пользователь кликает «Получить рекомендацию», используется запрос типа POST; соответственно, функция `handle_text_request` извлекает данные вопроса, передает их модели и возвращает ее вывод.



Какой вопрос вы хотели бы задать?

Это тестовый вопрос. Он хороший?

Получить рекомендацию

Вернуться на главную страницу

Рис. 9.2. Простая веб-страница для использования модели

Потоковое приложение следует использовать при наличии строгих требований к времени отклика. Если необходимая модели информация будет доступна только на этапе прогнозирования, а ответ требуется немедленно, то вам следует использовать потоковый подход. Например, для модели, прогнозирующей стоимость конкретной поездки на такси, нужна информация о местоположении пользователя и свободных водителях. Эти данные будут доступны только в момент поступления запроса. Предсказание такой модели должно немедленно отображаться пользователю.

В других ситуациях информация, необходимая для расчета предсказаний, может быть получена заранее. В таких случаях проще обрабатывать большое количество запросов за один раз, а не по мере их поступления. Давайте подробнее обсудим такое *пакетное предсказание*.

Пакетные предсказания

При пакетном подходе пайплайн инференса рассматривается как задание, которое может выполняться сразу для множества образцов. Пакетное задание запускает модель для множества образцов и сохраняет результаты для

использования по мере необходимости. Пакетный подход будет уместен, когда нужные модели признаки можно получить заранее, до момента выдачи предсказания.

Допустим, вам нужно создать модель, которая выдает маркетологам список самых перспективных потенциальных клиентов. Это распространенная задача машинного обучения, называемая *оценкой лидов*¹. Для обучения такой модели в качестве признаков можно использовать исторические данные о рыночных тенденциях и переписке по электронной почте. Эти признаки можно получить до момента получения предсказания, а именно до принятия решения о том, с кем из потенциальных клиентов следует связаться. Можно ночью ранжировать список лидов по степени готовности к покупке и получать готовые к отображению результаты к утру, когда в них возникает потребность.

Другой пример. Приложение, использующее МО для приоритизации электронных сообщений, которые пользователь получает каждое утро, тоже не предъявляет серьезных требований к времени отклика. Для этого приложения также подойдет пакетная схема обработки: каждое утро нужно партиями обрабатывать все непрочитанные электронные письма и сохранять полученный ранжированный список до момента, когда он понадобится пользователю.

Хотя при пакетном подходе требуется столько же раз вызвать инференс, сколько и при потоковом, он часто оказывается более ресурсоэффективным. Поскольку предсказания рассчитываются в заранее заданное время и вы знаете, сколько их будет при запуске каждой партии, гораздо легче выделить и распараллелить ресурсы. Кроме того, пакетный подход часто ускоряет работу на этапе инференса, так как вам требуется лишь извлечь ранее вычисленные результаты. Это дает тот же эффект, что и кэширование.

На рис. 9.3 показаны две составляющие этой схемы обработки. На этапе пакетной обработки мы вычисляем предсказания для всех элементов данных и сохраняем полученные результаты. На этапе инференса мы извлекаем ранее вычисленные результаты.

Также можно использовать комбинированный подход. Вычисляйте результат заранее в тех случаях, когда это возможно, и на этапе инференса либо извлекайте ранее вычисленные результаты, либо, если они отсутствуют или уже устарели, рассчитывайте их на месте. При таком подходе результаты будут получены максимально быстро, поскольку вы вычисляете все, что можно, заранее. За это приходится платить повышением сложности системы, так как нужно управлять одновременно и пакетным, и потоковым пайплайном.

Таким образом, мы рассмотрели два распространенных способа развертывания приложений на сервере — потоковый и пакетный. Оба подхода требуют, чтобы

¹ Лид — потенциальный клиент. — *Примеч. ред.*

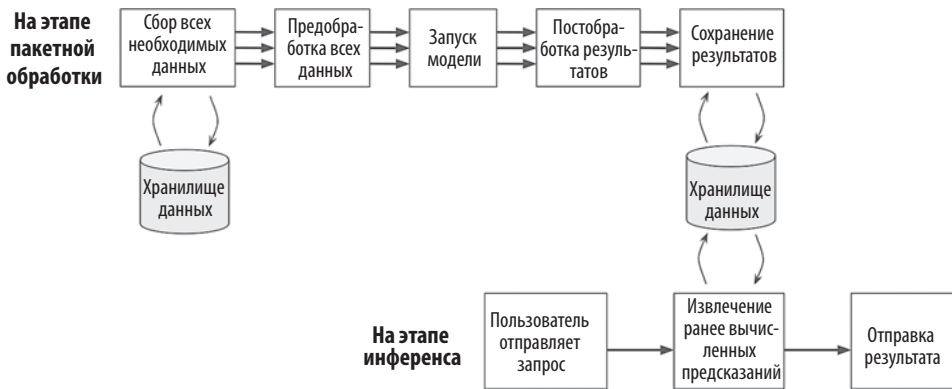


Рис. 9.3. Пример пакетной схемы обработки

инференс запускался на хостинговых серверах, что очень быстро приводит к большой трате ресурсов при росте популярности продукта. Серверы также становятся главным критическим узлом приложения, поскольку, если потребность в предсказаниях возрастет, они могут не справиться со всеми запросами.

В качестве альтернативы можно обрабатывать запросы непосредственно на тех клиентских устройствах, от которых они поступают. Запуск моделей на устройствах пользователей снижает затраты на инференс и позволяет поддерживать стабильный уровень сервиса вне зависимости от популярности приложения. Необходимые вычислительные ресурсы предоставляются клиентами. Такой подход называется *развертыванием на стороне клиента*.

Развертывание на стороне клиента

Цель развертывания моделей на стороне клиента — перенести все вычисления на клиентское устройство и тем самым исключить необходимость в запуске моделей на сервере. Современные персональные компьютеры, планшеты, смартфоны и другие устройства с выходом в интернет (наподобие «умных» колонок и домофонов) обладают достаточной вычислительной мощностью для запуска моделей.

Мы рассмотрим здесь только развертывание *обученных моделей* для инференса, без обучения моделей на устройстве. Модели по-прежнему обучаются на сервере, а затем переносятся на устройство для выполнения инференса. Модель может попасть на устройство либо в составе приложения, либо путем загрузки из веб-браузера. На рис. 9.4 показано, как выглядит данная схема обработки при включении модели в приложение.

Поскольку мобильные устройства обладают более скромными вычислительными ресурсами по сравнению с мощными серверами, данный подход накладывает ограничение на сложность используемых моделей. В то же время запуск моделей на устройстве имеет целый ряд преимуществ.

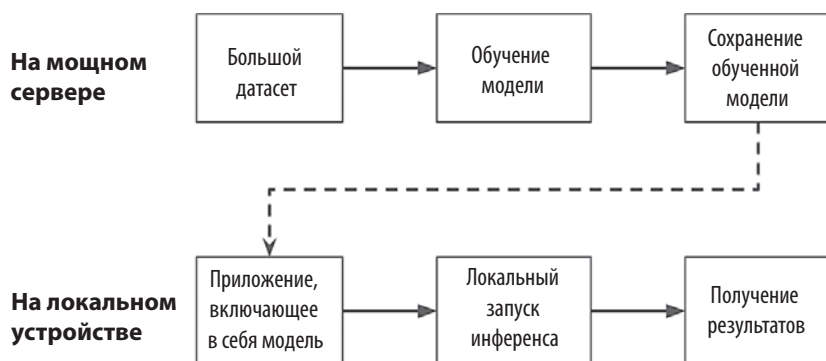


Рис. 9.4. Модель, выполняющая вывод на устройстве (обучение по-прежнему выполняется на сервере)

Во-первых, такой подход снижает потребность в создании инфраструктуры, обеспечивающей запуск инференса для каждого отдельного пользователя. Во-вторых, вам требуется пересылать меньше данных между устройством и сервером. Это ведет к уменьшению сетевой задержки и даже иногда позволяет использовать приложение без доступа к сети.

Наконец, если необходимые для инференса данные содержат конфиденциальную информацию, запуск модели на устройстве устраняет необходимость в передаче этих данных на удаленный сервер. В свою очередь, отсутствие конфиденциальных данных на серверах снижает риск незаконного доступа к ним посторонних лиц (почему это может представлять серьезную опасность, было рассказано в разделе «Забота о данных» на с. 204).

На рис. 9.5 показано, как предсказания выдаются пользователям в случае развертывания моделей на стороне сервера и на стороне клиента. Как видно из верхней схемы, большую часть времени в серверной схеме обработки занимает передача данных на сервер. При развертывании моделей на стороне клиента данные передаются практически без задержки, но в силу аппаратных ограничений снижается скорость обработки образцов (нижняя схема).

Как и в случае развертывания на сервере, существуют различные методы развертывания приложений на стороне клиента. В следующих разделах мы рассмотрим два метода — развертывание моделей в виде нативного приложения и их запуск посредством браузера. Эти методы хорошо подходят для смартфонов

и планшетов, имеющих доступ к магазину приложений и веб-браузеру, но не для других устройств с выходом в интернет, таких как микроконтроллеры, которых мы не будем касаться в этой книге.

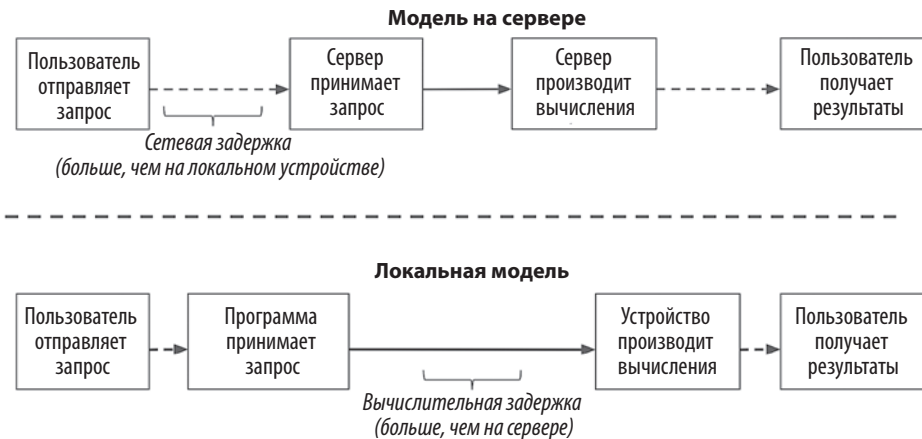


Рис. 9.5. Запуск на сервере в сравнении с локальным запуском

Развертывание на устройстве

Процессоры ноутбуков и смартфонов, как правило, не оптимизированы для запуска МО-моделей и потому выполняют пайплайн инференса медленнее. Чтобы развернутая на стороне клиента модель запускалась быстро и не требовала слишком много вычислительных ресурсов, следует использовать модели минимально возможного размера.

Уменьшить размер модели можно путем использования ее более простой версии, сокращения количества параметров или снижения точности расчетов. Например, в случае нейронных сетей это часто делается путем отсечения весов (удаления весов со значениями, близкими к нулю) или их квантизации (снижения их точности). Иногда можно получить дополнительный прирост производительности за счет уменьшения количества используемых моделью признаков. В последние годы такие библиотеки, как Tensorflow Lite¹, начали предлагать удобные инструменты для уменьшения размера моделей, чтобы их было проще развертывать на мобильных устройствах.

В силу указанных ограничений при переносе модели на устройство ее производительность обычно немного снижается. Если для вашего продукта это недопустимо (например, сложные ультрасовременные модели невозможно за-

¹ <https://oreil.ly/GKYDs>

пустить на таком устройстве, как смартфон), то придется развернуть модель на сервере. Общее правило сводится к следующему: если выполнение инференса на устройстве занимает больше времени, чем пересылка данных для их обработки на сервере, то следует запускать модель в облаке.

Для других приложений, например умной клавиатуры смартфонов, выдающей подсказки для ускорения ввода текста, польза от использования локальной модели, которая может работать без доступа к интернету, перевешивает сопутствующую потерю точности. Сходным образом приложение для смартфонов, позволяющее любителям пешего туризма опознать любое растение по фотографии, должно работать офлайн, чтобы его можно было использовать в походе. При создании такого приложения модель следует развернуть на устройстве, даже если за это придется заплатить некоторой потерей точности предсказаний.

Еще один пример МО-приложения, которому больше подойдет локальное развертывание, — программа для перевода. Такое приложение часто используется пользователями за границей, где они могут не иметь доступа к сети.

Помимо проблем со связью, развертывание моделей в облаке несет за собой риск нарушения конфиденциальности. Пересылка пользовательских данных в облако и сохранение их там даже на небольшой срок повышает риск того, что к ним получат доступ злоумышленники. В качестве примера давайте возьмем приложение, которое служит столь невинной цели, как редактирование фотографий путем наложения фильтров. Пользователи такого приложения вряд ли захотят, чтобы их фотографии пересылались на сервер для обработки и хранились там в течение неопределенного срока. Дав пользователям гарантию в том, что фотографии останутся на их устройстве, вы обеспечите себе серьезное конкурентное преимущество в силу того, что сегодня все больше внимания уделяется защите конфиденциальности. Как мы видели в разделе «Забота о данных» на с. 204, самый эффективный способ снизить риск незаконного доступа к конфиденциальным данным — не допустить пересылки этих данных за пределы устройства и сохранение их на серверах.

С другой стороны, квантизация, отсечение весов и упрощение модели требуют определенных затрат времени. Поэтому разворачивать модель на устройстве имеет смысл лишь в том случае, когда это оправдано временем отклика, инфраструктурой и конфиденциальностью. В случае МО-редактора мы ограничимся потоковым веб-API.

Наконец, следует иметь в виду, что оптимизация моделей под конкретные виды устройств может отнимать много времени, поскольку для разных устройств часто требуется разный процесс оптимизации. В таком случае лучше воспользоваться другими вариантами локального развертывания, которые позволяют сократить объем усилий за счет использования общих черт устройств. Большим потенциалом в этом плане обладает такой метод, как развертывание МО-моделей в браузере.

Развертывание в браузере

Большинство умных устройств имеют доступ к браузеру. Браузеры часто оптимизированы для быстрого выполнения графических вычислений. Отсюда — растущий интерес к библиотекам, использующим браузеры для выполнения задач МО на стороне клиента.

Наиболее популярным инструментом такого рода является фреймворк `Tensorflow.js`¹, который позволяет проводить обучение и выполнять инференс, используя JavaScript-код в браузере. Это работает для большинства моделей, даже тех, которые обучали с использованием кода на другом языке, например Python.

Пользователи при этом могут взаимодействовать с моделями через браузер, без установки каких-либо дополнительных приложений. Кроме того, поскольку модели запускаются в браузере с помощью JavaScript, вычисления производятся на устройстве пользователя. Ваша инфраструктура должна лишь доставлять веб-страницу, содержащую веса модели. Наконец, благодаря встроенной поддержке библиотеки `WebGL`, `Tensorflow.js` может повысить скорость вычислений, задействовав графический процессор клиентского устройства.

Использование JavaScript-фреймворка упрощает развертывание модели на стороне клиента, требуя гораздо меньше усилий по настройке под конкретное устройство по сравнению с предыдущим подходом. Недостаток этого подхода — более высокие требования к трафику, поскольку модель должна загружаться клиентом при каждом открытии страницы, а не только при установке приложения.

Если размер используемых вами моделей не больше нескольких мегабайтов и они быстро загружаются, их запуск на клиенте с помощью JavaScript-кода может быть удобным способом снижения нагрузки на сервер. Если нагрузка на сервер станет проблемой для нашего МО-редактора, то я порекомендовал бы развернуть модели с использованием фреймворка `Tensorflow.js`.

До сих пор мы рассматривали лишь возможность развертывания на стороне клиента уже обученной модели, однако иногда бывает полезно и обучать модели на клиенте. В оставшейся части главы мы разберемся, в каких случаях такой подход может быть целесообразным.

Федеративное обучение: комбинированный подход

До сих пор мы говорили о том, как выбрать способ развертывания модели в том случае, когда она уже обучена (желательно в соответствии с рекомен-

¹ <https://www.tensorflow.org/js>

дациями, изложенными в предыдущих главах). Мы рассмотрели различные способы предоставления всем пользователям одной и той же модели. А если мы захотим, чтобы у каждого пользователя была своя модель, отличающаяся от остальных?

Рисунок 9.6 показывает разницу между одной общей моделью для всех пользователей и привлечением множества слегка различающихся версий модели.

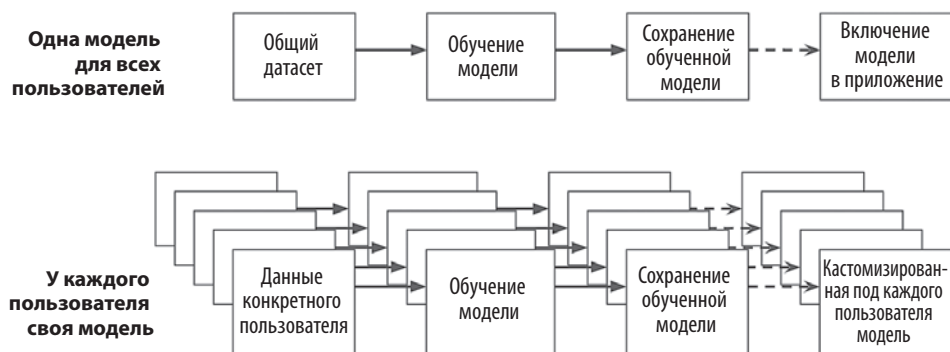


Рис. 9.6. Одна большая модель или множество отдельных моделей

Для многих программ, таких как приложения по рекомендации контента, помощники по написанию текстов или медицинские приложения, наиболее важным источником информации для модели являются данные о пользователе. Мы можем либо создавать для модели признаки, отражающие особенности каждого пользователя, либо предоставлять каждому пользователю собственную модель. Модели могут иметь одинаковую архитектуру, но разные значения параметров, отражающие индивидуальные данные отдельных пользователей.

Эта идея лежит в основе федеративного обучения — разновидности глубокого обучения, которая притягивает к себе все больше внимания после появления таких проектов, как OpenMined¹. Метод федеративного обучения сводится к следующему. Каждому клиенту предоставляется собственная модель. Каждая модель обучается на данных отдельного пользователя и отправляет на сервер агрегированные (и анонимизированные) обновления. Сервер использует все полученные обновления для улучшения модели, после чего возвращает клиентам новую модель.

Модель каждого клиента адаптируется под его персональные потребности и в то же время извлекает определенную пользу из агрегатной информации

¹ <https://www.openmined.org/>

обо всех остальных клиентах. Федеративное обучение позволяет сохранить конфиденциальность, поскольку данные отдельных пользователей никогда не передаются на сервер, который получает только агрегированные обновления модели. Это существенно отличается от традиционного способа обучения модели, при котором нужно собирать и сохранять на сервере данные о каждом пользователе.

Федеративное обучение является очень перспективным направлением развития МО, но в то же время вносит дополнительную сложность. Вы должны проследить за тем, чтобы каждая отдельная модель работала достаточно эффективно и чтобы должным образом анонимизировались передаваемые на сервер данные, что гораздо сложнее по сравнению с обучением одной модели.

Федеративное обучение уже применяется в реальных приложениях командами разработчиков, располагающими достаточными ресурсами для его развертывания. Например, в статье А. Харда (A. Hard) и др. «Федеративное обучение для прогнозирования ввода мобильной клавиатуры» («Federated Learning for Mobile Keyboard Prediction»)¹ описывается опыт применения федеративного обучения в мобильном приложении Google Gboard для прогнозирования следующего вводимого слова. Тот факт, что разные пользователи могут использовать разный стиль написания текстов, усложняет задачу создания единой модели, способной эффективно работать для каждого пользователя. Обучение моделей на уровне отдельных пользователей позволяет приложению GBoard выявлять особенности их поведения и благодаря этому делать более удачные прогнозы.

Таким образом, мы рассмотрели различные способы развертывания моделей на сервере, на устройствах, а также на том и другом одновременно. Вы должны взвесить плюсы и минусы каждого подхода с учетом требований, которые предъявляются к вашему приложению. Как и в других главах этой книги, я рекомендую начать с простейшего подхода и переходить к более сложному, лишь убедившись в том, что это действительно необходимо.

Заключение

Существуют разные способы поставки приложения на базе МО. Вы можете построить потоковый API так, чтобы модель могла обрабатывать образцы по мере их поступления. Вы можете использовать пакетную схему работы, обрабатывая сразу большое количество образцов с некоторой установленной периодичностью. Вы также можете развертывать свои модели на стороне клиента, включая их в состав приложения либо доставляя посредством веб-браузера. Такой под-

¹ <https://arxiv.org/abs/1811.03604>

ход снижает затраты ресурсов на инференс и инфраструктуру, но несколько усложняет процесс развертывания.

Выбор подхода зависит от того, какие требования предъявляются к вашему приложению в плане времени отклика, аппаратного обеспечения, использования сети, защиты конфиденциальности, ресурсов на вывод и т. д. В случае такого простого прототипа, как наш МО-редактор, следует начать с использования эндпоинтов или простейшего пакетного подхода и при необходимости доработать его.

Однако развертывание модели не ограничивается лишь тем, чтобы сделать ее доступной пользователям. В главе 10 мы рассмотрим способы создания защитных механизмов, снижающих вероятность появления ошибок, инструменты разработчика для эффективного развертывания и методы подтверждения того, что модель работает правильно.

Создание защитных механизмов для моделей

При проектировании баз данных или распределенных систем необходимо обеспечить отказоустойчивость, то есть способность системы сохранять работоспособность при отказе отдельных компонентов. В случае программного обеспечения вопрос заключается не в том, может ли отказать та или иная часть системы, а в том, когда это произойдет. То же самое справедливо и для приложений на базе МО. Даже очень хорошая модель может давать сбой на некоторых образцах, поэтому вы должны спроектировать свою систему так, чтобы она не теряла работоспособность в случае такого сбоя.

В этой главе мы рассмотрим различные способы предотвращения или снижения вероятности ошибок. Сначала мы узнаем, как следует проверять качество входных и выходных данных и изменять способ отображения выходных данных в зависимости от результата этой проверки. Затем рассмотрим способы повышения устойчивости пайплайна моделирования, чтобы эффективно обслуживать большое количество пользователей. После этого поговорим о различных способах получения обратной связи и оценки состояния модели. Завершит главу интервью с Крисом Муди (Chris Moody), в котором он поделится своими рекомендациями по развертыванию.

Проектирование с учетом возможных сбоев

Давайте рассмотрим некоторые распространенные случаи сбоев пайплайна МО. Внимательные читатели могли заметить, что мы уже обсуждали примерно то же самое в разделе «Отладка потока данных: визуализация и тестирование» на с. 160. Действительно, при предоставлении пользователям доступа к модели в реальных условиях приходится иметь дело практически с теми же проблемами, что и в случае отладки модели.

Хотя ошибки могут возникать где угодно, особенно важно проверить следующие три области: поступающие в пайплайн входные данные, предсказания модели и их вероятности. Давайте посмотрим, как это сделать.

Проверка входных данных и результатов

Любая модель обучается на конкретном датасете с конкретными характеристиками. Обучающие данные обладают некоторым количеством признаков разного типа. Кроме того, каждый признак каким-то образом распределен, и для получения точных результатов модель должна «выучить» характер этого распределения.

Как мы видели в разделе «Актуальность данных и сдвиг распределения» на с. 50, модель может показывать плохие результаты, когда в реальных условиях данные отличаются от тех, на которых проводилось обучение. Для решения этой проблемы вы должны проверять входные данные, поступающие в ваш пайплайн.

Проверка входных данных

Некоторые модели показывают хорошие результаты даже при наличии небольших отличий в распределении данных. Однако ни одна модель не может хорошо работать, когда входные данные сильно отличаются от обучающих, отсутствуют некоторые признаки или часть признаков имеют неожиданный тип.

Как мы видели ранее, МО-модели способны работать даже при получении некорректных входных данных (при условии, что данные обладают надлежащим типом и форматом). Модель все равно выдает результаты, но эти результаты могут быть совершенно неправильными. Давайте рассмотрим пример, показанный на рис. 10.1. Данный пайплайн относит предложение к одной из двух тем. Сначала предложение векторизуется, а затем к векторному представлению применяется классификационная модель. Если на вход такого пайплайна поступит строка из случайной последовательности символов, то она, как обычно, преобразуется в вектор, и модель выдаст некоторое предсказание. Хотя предсказание будет совершенно абсурдным, вы не сможете этого заметить, если будете смотреть только на результаты модели.

Чтобы не запускать модель на некорректных входных данных, мы должны распознавать их до передачи модели.

Проверки входных данных решают почти те же задачи, что и тесты, упоминаемые в разделе «Тестирование МО-кода» на с. 167. В порядке убывания важности необходимо проверить следующее.

1. Наличие всех необходимых признаков.
2. Корректность типа каждого признака.
3. Корректность значений признаков.

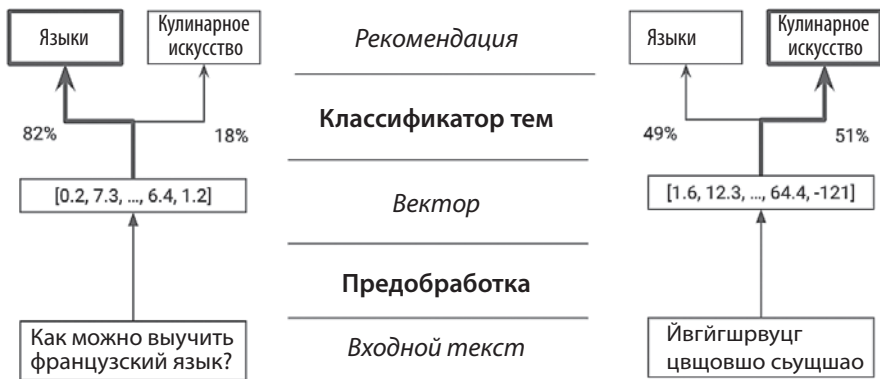


Рис. 10.1. Модель выдает предсказание даже при поступлении на вход случайной последовательности символов

ОТЛИЧИЕ ПРОВЕРОК ОТ ТЕСТОВ

В этом разделе мы говорим о проверках входных данных. Это следует отличать от тестов, о которых мы рассказывали в разделе «Тестирование МО-кода» на с. 167. Между первым и вторым имеется тонкое, но важное различие. Тесты служат для подтверждения того, что код ведет себя ожидаемым образом при передаче ему некоторых известных и заранее определенных входных данных. Тесты обычно запускают после каждого изменения кода или модели, чтобы убедиться в том, что пайплайн по-прежнему работает должным образом. А рассматриваемые в данном разделе проверки являются частью самого пайплайна и определяют порядок выполнения программы в зависимости от качества входных данных. Если входные данные не пройдут определенную проверку, то может запускаться другая модель или вообще никакая.

Проверка корректности значений признаков может сама по себе вызывать трудности, поскольку распределение признаков часто бывает непростым. Простейший способ сводится к тому, чтобы определить диапазон допустимых значений и проверить, находится ли значение признака в пределах этого диапазона.

Вы не должны запускать модель, если не будет пройдена какая-либо из входных проверок. Что именно нужно будет при этом сделать, зависит от юзкейса. Так, при отсутствии некоторого важного элемента данных вы должны выдать сообщение с указанием источника ошибки. Если вы понимаете, что даже в таком случае можно получить некоторый результат, то вызов модели можно заменить вызовом эвристического алгоритма. Это еще один довод в пользу того, чтобы начинать работу над любым проектом МО с реализации эвристического алгоритма, — так вы сможете всегда иметь под рукой запасной вариант.

На рис. 10.2 показано, как может выглядеть логическая схема в зависимости от результата входных проверок.

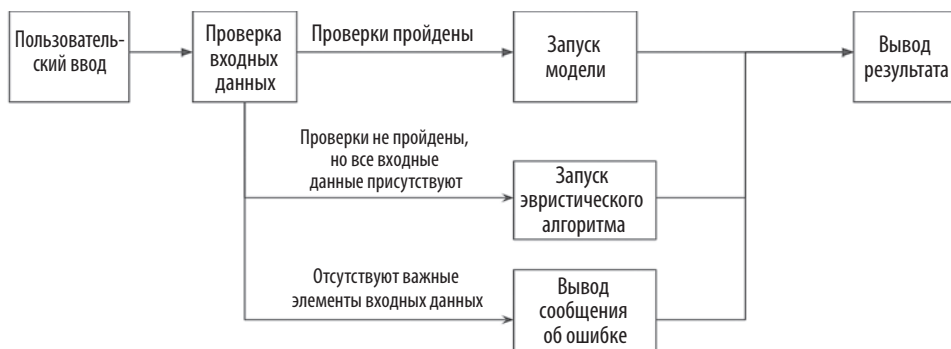


Рис. 10.2. Пример логики для выполнения входных проверок

Ниже в качестве примера приводится часть управляющей логики МО-редактора, которая проверяет наличие признаков и корректность их типа. В зависимости от качества входных данных, код либо выдает сообщение об ошибке, либо выполняет эвристический алгоритм. Как и остальной код МО-редактора, этот пример можно найти в GitHub-репозитории книги¹.

```

def validate_and_handle_request(question_data):
    missing = find_absent_features(question_data)
    if len(missing) > 0:
        raise ValueError("Missing feature(s) %s" % missing)

    wrong_types = check_feature_types(question_data)
    if len(wrong_types) > 0:
        # Запускаем эвристический алгоритм, если данные некорректны,
        # но известна длина вопроса
        if "text_len" in question_data.keys():
            if isinstance(question_data["text_len"], float):
                return run_heuristic(question_data["text_len"])
            raise ValueError("Incorrect type(s) %s" % wrong_types)

    return run_model(question_data)
  
```

Проверка входных данных модели позволяет локализовать возможные виды ошибок и распознать проблемы с вводом данных. После выполнения этой проверки вы также должны проверить предсказания модели.

¹ <https://github.com/hundredblocks/ml-powered-applications>

Вывод модели

Когда модель выдает предсказание, вы должны решить, следует ли отображать его пользователю. Отображать не следует, если оно выходит за пределы допустимого диапазона значений.

Например, если вы пытаетесь определить возраст пользователя по его фотографии, то результат должен находиться в диапазоне от нуля до ста с небольшим лет (можете подправить границы, если вы читаете эту книгу в 3000 году). Вы не должны отображать результат модели, если он выходит за пределы этого диапазона.

Приемлемость результата определяется не только степенью его правдоподобности, но и вашими представлениями о том, каким должен быть *полезный для пользователя* результат.

В случае нашего МО-редактора нужно лишь предоставить некоторые практически полезные рекомендации. Если модель сообщит, что пользователь должен удалить весь ранее написанный текст, это будет малополезная (и даже оскорбительная) рекомендация. Представленный ниже фрагмент кода демонстрирует, как можно проверить результаты модели и при необходимости откатиться к эвристическому алгоритму:

```
def validate_and_correct_output(question_data, model_output):
    # Проверяем тип и диапазон и выдаем соответствующее сообщение об ошибках
    try:
        # Вызываем ошибку значения, если модель выдает некорректный результат
        verify_output_type_and_range(model_output)
    except ValueError:
        # Запускаем эвристический алгоритм; здесь также можно
        # запустить другую модель
        run_heuristic(question_data["text_len"])

    # При отсутствии ошибок возвращаем результат модели
    return model_output
```

В случае сбоя модели вы можете откатиться к эвристическому алгоритму, как было показано ранее, или воспользоваться имеющимся у вас более простым вариантом модели. Бывает полезно применить предыдущий вариант модели, поскольку разные модели могут совершать непохожие ошибки.

Иллюстрацией этому может служить небольшой пример, представленный на рис. 10.3. Слева показаны результаты более производительной модели, граница принятия решений которой имеет более сложную конфигурацию, справа — результаты менее производительной и более простой модели. Менее эффективная модель делает больше ошибок, но эти ошибки не совпадают с ошибками сложной модели, поскольку ее решающая граница выглядит иначе. В силу этого простая модель выдает правильный результат для некоторых образцов, с которыми не

может справиться сложная модель. Именно поэтому разумно иметь простую модель в качестве резервного варианта на случай сбоя основной модели.

Если вы решите использовать более простую модель в качестве резервного варианта, вам потребуется точно так же проверять ее результаты и далее либо делать откат к эвристическому алгоритму, либо отображать сообщение об ошибке при непрохождении проверок.

Хотя проверка результатов модели на принадлежность к диапазону допустимых значений будет хорошей отправной точкой, вы не можете этим ограничиться. В следующем разделе мы поговорим о том, какими еще защитными механизмами следует снабдить модель.



Рис. 10.3. Более простая модель часто делает другие ошибки

Резервные варианты на случай сбоя модели

Итак, мы создали защитные механизмы для выявления и исправления ошибочных входных и выходных данных. Однако иногда, при вполне корректных входных данных, модель может выдавать разумные, но в действительности совершенно неверные предсказания.

Давайте вернемся к примеру с определением возраста пользователя по его фотографии. Если модель гарантированно выдает правдоподобный для человека возраст, это хорошая отправная точка, но в идеале она должна правильно определять возраст конкретного пользователя.

Ни одна модель не может работать совсем без ошибок, и наличие небольших погрешностей вполне допустимо. В то же время вы должны по возможности выявлять, где модель работает неправильно. Это позволит вам пометить опре-

деленные случаи как слишком сложные, тем самым призывая пользователей предоставлять легко обрабатываемые входные данные (например, фотографии, снятые в условиях хорошей освещенности).

Существуют два основных подхода к выявлению ошибок. Простейший подход — оценивать точность результата по уверенности модели в предсказании. Второй подход сводится к тому, чтобы использовать дополнительную модель для выявления тех образцов, на которых может дать сбой основная модель.

При использовании первого способа в качестве оценки уверенности модели можно применить вычисляемый классификатором вероятностный показатель. Если вероятностный показатель хорошо откалиброван (см. раздел «Калибровочная кривая» на с. 145), с его помощью можно определить, когда модель не уверена в предсказании, и не отображать пользователю результат.

Иногда модель выдает неверный результат, невзирая на то что соответствующему образцу присваивается высокий вероятностный показатель. Именно здесь вступает в игру второй подход: используя дополнительную модель, вы можете отфильтровать те входные данные, которые вызывают проблемы.

Фильтрующая модель

Помимо недостаточной надежности, использование показателя уверенности модели обладает еще одним серьезным недостатком. Чтобы получить этот показатель, вы должны выполнить весь пайплайн инференса вне зависимости от того, понадобится ли соответствующее предсказание. Это особенно нерационально, когда вы используете достаточно сложную модель, для запуска которой требуется GPU. В идеале оценку того, насколько хороший результат может выдать модель для конкретного образца, нужно получать без запуска модели.

Именно в этом заключается смысл фильтрующей модели. Поскольку вы знаете, что модели будет трудно справиться с некоторой частью входных данных, вы должны выявить их заранее, вообще не запуская при этом модель. Фильтрующая модель — это МО-версия входных тестов. Это бинарный классификатор, обученный предсказывать, может ли основная модель выдать хороший результат для заданного образца. Модель основана на предположении, что в элементах данных, представляющих трудности для основной модели, имеются некоторые закономерности. Если такие проблемные образцы имеют достаточно много общего, то фильтрующая модель может научиться отделять их от легко обрабатываемых входных данных.

Помимо прочего, фильтрующая модель может перехватывать следующие разновидности входных данных:

- Входные данные, которые качественно отличаются от тех, на которых основная модель показывает хорошие результаты.

- Входные данные, которые использовались для обучения модели, но представляли трудности для нее.
- Вредоносные входные данные, созданные с целью обмануть основную модель.

На рис. 10.4 показана обновленная версия логики с рис. 10.2, которую мы дополнили фильтрующей моделью. Как видим, фильтрующая модель запускается только в случае успешного прохождения входных проверок, поскольку фильтроваться должны лишь те входные данные, которые способны дойти до пункта «Запуск модели».

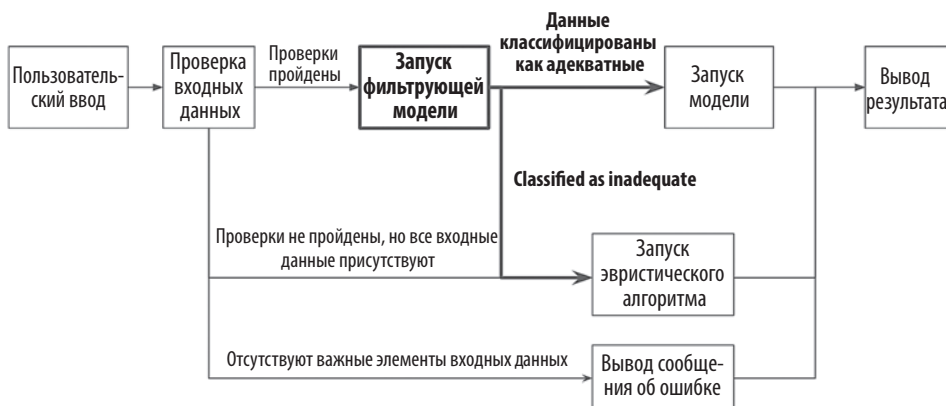


Рис. 10.4. Добавление фильтрации входных данных (выделено полужирным шрифтом)

Чтобы обучить фильтрующую модель, вам нужно просто сформировать датасет, содержащий две категории образцов: образцы, на которых основная модель показывает хорошие результаты, и образцы, на которых она дает сбой. Это можно сделать с помощью имеющихся обучающих данных, собирать дополнительные данные не потребуется!

На рис. 10.5 продемонстрировано, как это можно сделать, используя обученную модель. Слева на графике — результаты ее работы. Отберите несколько элементов данных, для которых модель выдает правильный результат, и несколько элементов, на которых она ошибается. После этого можно обучить фильтрующую модель выявлять такие проблемные элементы.

После того как вам удастся обучить классификатор, создание фильтрующей модели уже не будет представлять большого труда. При наличии тестового набора и обученного классификатора это можно сделать с помощью следующей функции:

```
def get_filtering_model(classifier, features, labels):
    """
```

Выявляет ошибки предсказания для датасета бинарной классификации

```

:param classifier: обученный классификатор
:param features: входные признаки
:param labels: реальные метки
"""
predictions = classifier.predict(features)
# Создаем метки, равные 1 в случае ошибки и 0 в случае правильного
# предсказания
is_error = [pred != truth for pred, truth in zip(predictions, labels)]

filtering_model = RandomForestClassifier()
filtering_model.fit(features, is_error)
return filtering_model

```

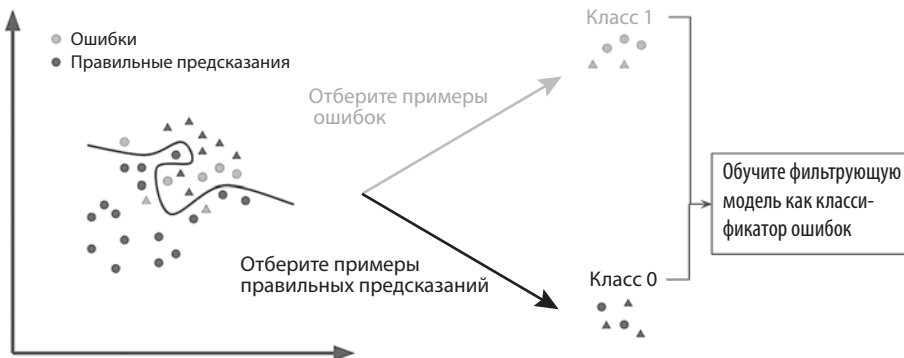


Рис. 10.5. Получение обучающих данных для фильтрующей модели

Компания Google применяет этот подход для функции «умного ответа» Smart Reply, которая предлагает несколько коротких вариантов ответа на входящее электронное письмо (см. статью А. Канан (A. Kanan) и др. «Smart Reply: предложение по автоматическому ответу на электронное письмо» («Smart Reply: Automated Response Suggestion for Email»¹)). При этом используется так называемая модель активации, которая решает, следует ли запускать основную модель, предлагающую варианты ответов. Как оказалось, для такой модели пригодны примерно 11% электронных писем. Таким образом, использование фильтрующей модели позволило на порядок уменьшить потребность в инфраструктуре.

В большинстве случаев фильтрующая модель должна удовлетворять двум критериям: быть быстрой, поскольку используется лишь для снижения затрат вычислительных ресурсов, и хорошо справляться с устранением тяжелых случаев.

При этом не нужно, чтобы фильтрующая модель выявляла буквально все тяжелые случаи; достаточно, что она будет оправдывать дополнительные затраты на

¹ <https://oreil.ly/2EQvu>

свой запуск при каждой операции вывода. Чем быстрее работает фильтрующая модель, тем ниже требования к ее производительности. Можно объяснить это так:

Допустим, что при использовании только основной модели средняя длительность операции вывода равна i .

При наличии фильтрующей модели средняя длительность вывода будет равна $f + i(1 - b)$, где f — время работы фильтрующей модели, а b — средняя доля отсеиваемых образцов (от слова «block» — «блокировать»).

Таким образом, чтобы использование фильтрующей модели вело к уменьшению средней длительности вывода, должно выполняться неравенство $f + i(1 - b) < i$, или после преобразования $f/i < b$.

То есть доля отсеиваемых случаев должна быть больше, чем соотношение между скоростью работы фильтрующей и основной модели.

Так, например, если фильтрующая модель работает в 20 раз быстрее, чем основная ($f/i = 5\%$), то она должна отсеивать как минимум 5% случаев ($5\% < b$), чтобы быть полезной в эксплуатационном окружении.

Конечно, при этом нужно убедиться в том, что фильтрующая модель работает достаточно точно, то есть большинство отсеиваемых ею входных данных действительно вызывает затруднения у основной модели.

Один из способов — периодически отбирать по несколько отсеиваемых образцов и проверять, как с ними может справиться основная модель. Мы обсудим это подробнее в разделе «Что мониторить» на с. 257.

Поскольку фильтрующая модель отличается от модели для инференса и специально обучается выявлению тяжелых случаев, она может быть более эффективной, чем использование вероятностного показателя основной модели. Таким образом, фильтрующая модель не только снижает вероятность выдачи плохих результатов, но и повышает эффективность использования ресурсов.

Добавление фильтрующей модели к имеющимся проверкам входных данных и результатов может существенно повысить надежность пайплайна в эксплуатационном окружении. В следующем разделе мы рассмотрим ряд других способов повышения надежности пайплайна по мере обсуждения того, как масштабировать МО-приложения при возрастании количества пользователей и как подходить к организации сложных процессов обучения.

Проектирование для обеспечения высокой производительности

Достаточно серьезной проблемой при развертывании моделей в эксплуатационном окружении является поддержка их производительности, особенно когда

продукт становится популярным и вам требуется регулярно разворачивать новые версии модели. Мы начнем этот раздел с обсуждения методов, позволяющих моделям обрабатывать большое количество запросов на инференс. Затем поговорим о том, как упростить регулярное развертывание обновленных версий модели. Наконец, обсудим, как уменьшить разброс производительности моделей, сделав более воспроизводимым пайплайн обучения.

Масштабирование при возрастании числа пользователей

Рабочая нагрузка программного обеспечения часто является масштабируемой по горизонтали. В таком случае сохранить приемлемое время отклика при росте числа запросов можно просто путем развертывания дополнительных серверов. Приложения на базе МО ничем не отличаются в этом плане, поскольку мы можем повышать производительность своих моделей, просто разворачивая дополнительные серверы для их запуска.



При использовании модели глубокого обучения часто требуется задействовать графический процессор (GPU), чтобы получать предсказания за приемлемое время. Если вы работаете с такой моделью и понимаете, что с ожидаемым количеством запросов нельзя будет справиться, используя GPU только одной машины, то вам следует запускать логику приложения и модель для инференса на двух разных серверах.

Поскольку у облачных провайдеров стоимость серверов с GPU обычно на порядок выше стоимости обычных облачных серверов, вы можете существенно снизить затраты на вычисления, если более дешевый из них будет использоваться для масштабирования приложения, а сервер с GPU — исключительно для инференса. Учитывая, что при использовании этой стратегии придется тратить больше ресурсов на обмен данными, убедитесь, что это для вас не проблема.

Помимо увеличения количества выделяемых ресурсов, приложения на базе МО также могут применять такие эффективные способы обработки дополнительного трафика, как кэширование.

Кэширование для МО

Под кэшированием понимается сохранение результатов вызова функции с целью более быстрого выполнения ее последующих вызовов с теми же параметрами за счет простого извлечения сохраненных результатов. Это распространенная практика для ускорения проектируемых пайплайнов, которая очень полезна и в сфере МО.

Кэширование результатов вывода. Одним из самых простых методов кэширования является вытеснение давно не используемых данных (Least Recently

Used, LRU). Данный метод сводится к отслеживанию самых свежих входных данных модели и соответствующих им предсказаний. Перед запуском модели для любого нового входного образца производится его поиск в кэше. При обнаружении соответствующей записи результат просто извлекается из кэша. Пример такой схемы обработки представлен на рис. 10.6. Сверху на рисунке показано, как производится кэширование при первом поступлении некоторого входного образца. Снизу показано, как извлекаются данные при повторном поступлении того же входного образца.

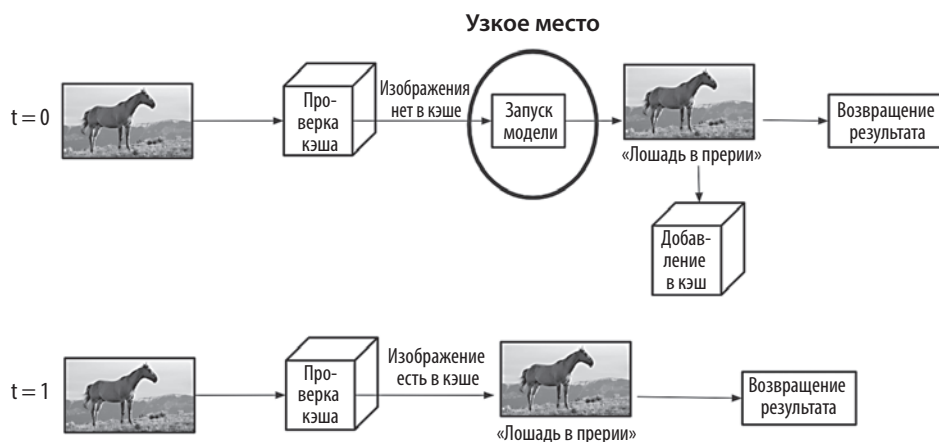


Рис. 10.6. Кэширование для модели генерирования описаний изображений

Такая стратегия кэширования хорошо подходит для приложений, пользователи которых могут предоставлять одинаковые входные данные, и будет неуместной, если входные данные никогда не повторяются. Так, в случае приложения для определения животного по фотографии его следов, сделанная пользователем фотография будет очень редко совпадать с предоставленными ранее, поэтому здесь нет смысла использовать LRU-кэширование.

Кэшировать необходимо только функции без побочных эффектов. Так, например, если функция `run_model` дополнительно сохраняет результаты в базе данных, использование LRU-кэша приведет к тому, что результаты повторных вызовов этой функции не будут сохраняться, что будет отклонением от нормального поведения.

В Python модуль `functools`¹ предлагает дефолтную реализацию LRU-кэша, которую можно использовать с помощью простого декоратора, как показано ниже:

¹ <https://oreil.ly/B73Bo>

```
from functools import lru_cache

@lru_cache(maxsize=128)
def run_model(question_data):
    # Ниже можно вставить любой вывод медленной модели
    pass
```

Кэширование полезно в случае, когда извлечение признаков, их обработка и выполнение инференса занимают больше времени, чем извлечение данных из кэша. Польза зависит от того, где вы сохраняете кэш (например, в оперативной памяти или на жестком диске), и от того, насколько сложной будет ваша модель.

Кэширование через индексацию. Хотя описанный выше метод кэширования не подходит, если входные данные никогда не повторяются, мы можем кэшировать другие составляющие пайплайна, которые могут вычисляться заранее. Проще всего, если модель полагается не только на пользовательские входные данные.

Допустим, нам нужно создать систему для поиска контента по предоставленной пользователем текстовой строке или фотографии. Если поисковые запросы будут сильно варьироваться, то их кэширование вряд ли даст значительный прирост производительности. Однако, поскольку мы имеем дело с поисковой системой, у нас есть доступ к списку элементов каталога, возвращаемых в ответ на запрос. Этот список будет известен заранее и в случае интернет-магазина, и в случае платформы для индексации документов.

Это означает, что мы можем заранее вычислять те составляющие моделирования, которые зависят только от элементов каталога. Выбрав метод моделирования, позволяющий производить эти вычисления заранее, мы сможем существенно ускорить инференс.

В силу вышесказанного распространенный подход при создании поисковой системы — вложить все индексируемые документы в информативный вектор (подробное описание методов векторизации см. в разделе «Векторизация» на с. 95). Затем вы можете сохранить эти эмбединги в базе данных. Схема процесса показана сверху на рис. 10.7. Поисковый запрос пользователя векторизуется на этапе инференса, после чего в базе данных производится поиск наиболее похожих эмбедингов и возвращаются соответствующие им объекты, как это показано снизу на рис. 10.7.

Такой подход существенно ускоряет выполнение инференса, поскольку большая часть расчетов выполняется заранее. Эмбединги с успехом используются в масштабных проектах таких компаний, как Twitter (см. соответствующий пост¹ в социальной сети Twitter) и Airbnb (см. статью М. Халдара (M. Haldar) и др.

¹ <https://oreil.ly/3R5hL>

«Применение глубокого обучения к поиску в Airbnb» («Applying Deep Learning To Airbnb Search»¹)).

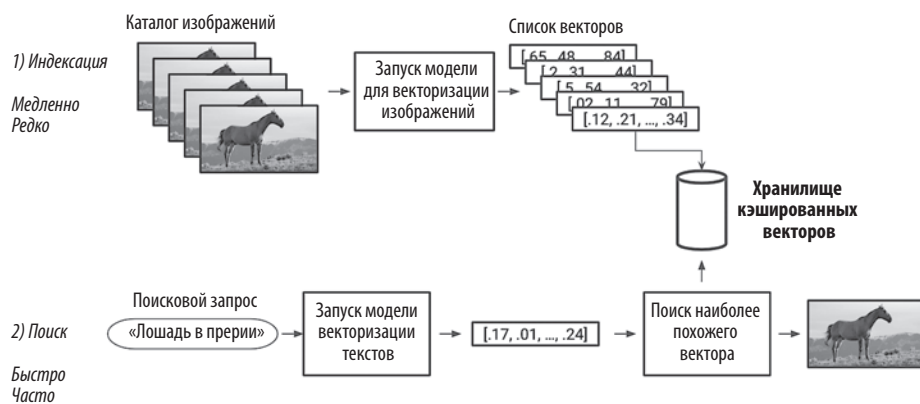


Рис. 10.7. Поисковый запрос с использованием кэшированных эмбеддингов

Повышая производительность, кэширование в то же время вносит некоторую сложность. Размер кэша становится дополнительным гиперпараметром, который нужно настраивать в зависимости от рабочей нагрузки приложения. Кроме того, вы должны очищать кэш при каждом обновлении модели или основных данных, чтобы не извлекать устаревшие результаты. В целом обновление модели в эксплуатационном окружении требует большой осторожности. В следующем разделе мы поговорим о том, как можно упростить выполнение таких обновлений.

Управление жизненным циклом модели и данных

Поддержание кэша и моделей в актуальном состоянии является непростой задачей. Для поддержания хорошей производительности многие модели нужно регулярно дообучать. В главе 11 мы еще обсудим, когда дообучать модели, а пока давайте кратко остановимся на том, как развернуть обновленные модели для пользователей.

Обычно обученная модель сохраняется в виде бинарного файла с информацией о ее типе, архитектуре и параметрах. Развернутое в эксплуатационном окружении приложение, как правило, загружает обученную модель в оперативную память в момент своего запуска и потом вызывает ее для выдачи результатов.

¹ <https://arxiv.org/abs/1810.09591>

Самый простой способ замены модели на более новую версию — заменить загружаемый приложением бинарный файл. Этот процесс показан на рис. 10.8: замена модели отражается только на одном элементе пайплайна, который выделен здесь полужирной рамкой.

ОБУЧЕНИЕ



ИНФЕРЕНС

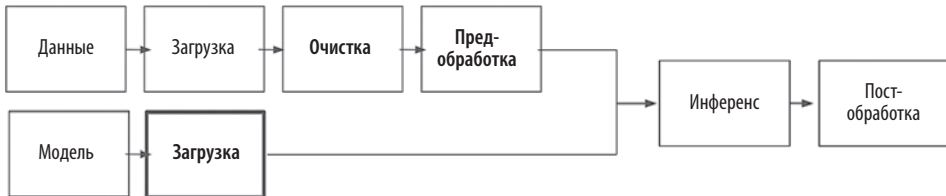


Рис. 10.8. Развертывание обновленной версии модели изначально кажется тривиальной задачей

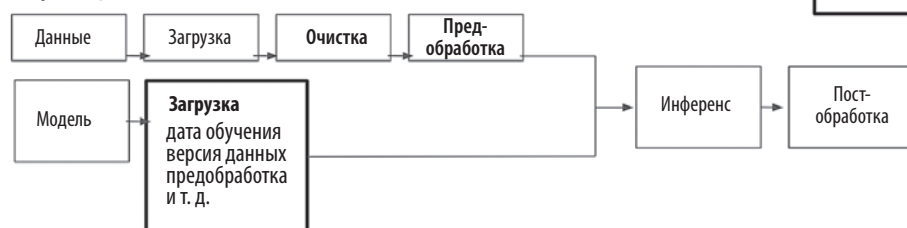
Однако на практике процесс далеко не так прост. В идеале МО-приложение должно выдавать воспроизводимые результаты, быть устойчивым к обновлению модели и достаточно гибким для того, чтобы можно было вносить значительные изменения в схему моделирования и обработки данных. Чтобы гарантировать это, потребуется выполнить несколько дополнительных шагов, о которых мы и поговорим далее.

Воспроизводимость

Чтобы отслеживать и воспроизводить ошибки, вы должны знать, какая модель работает в эксплуатационном окружении. Для этого надо сохранять архив, содержащий обученные модели и соответствующие датасеты. Каждой паре «модель — датасет» следует присвоить уникальный идентификатор и регистрировать его в логах при каждом использовании модели в эксплуатационном окружении.

На рис. 10.9 я добавил эти требования к полям загрузки и сохранения, чтобы показать, насколько это усложняет пайплайн машинного обучения.

Помимо предоставления пользователям различных версий имеющихся моделей, производственный пайплайн должен обновлять модели без значительных простоев.

ОБУЧЕНИЕ**ИНФЕРЕНС****Рис. 10.9.** Добавляем важные метаданные на этапе сохранения и загрузки**Устойчивость**

Чтобы приложение могло загрузить новую модель после каждого обновления, необходимо построить процесс загрузки обновленной модели. Лучше всего сделать это так, чтобы обслуживание пользователей не прерывалось. Можно просто запускать новый сервер для предоставления обновленной модели и постепенно переводить на него трафик, однако этот процесс быстро усложняется при возрастании размера системы. Также желательно предусмотреть возможность отката к предыдущей модели, если новая модель будет работать плохо. Хорошо реализовать обе задачи не так просто, и обычно этим занимаются DevOps-инженеры. В главе 11 мы в общих чертах рассмотрим, как проводить мониторинг, хотя не будем глубоко погружаться в эту тему.

Изменения в эксплуатационном окружении не ограничиваются простым обновлением модели. Они могут включать в себя значительные преобразования в обработке данных, которые также должны быть развертываемыми.

Гибкость пайплайна

Мы убедились, что самый эффективный способ улучшения модели часто сводится к корректировке процесса обработки данных и генерирования признаков. Это означает, что для новых версий модели требуются дополнительные операции предобработки или дополнительные признаки.

Такое изменение затрагивает не только двоичный файл модели, оно часто связано к новой версии приложения. Следовательно, надо логировать версию приложения при выдаче моделью предсказания, чтобы сделать его воспроизводимым.

Это вносит еще один уровень сложности в наш пайплайн: как видно из рис. 10.10, его схема теперь содержит дополнительные пункты предобработки и постобработки. Они тоже должны быть воспроизводимыми и модифицируемыми.

ОБУЧЕНИЕ



ИНФЕРЕНС

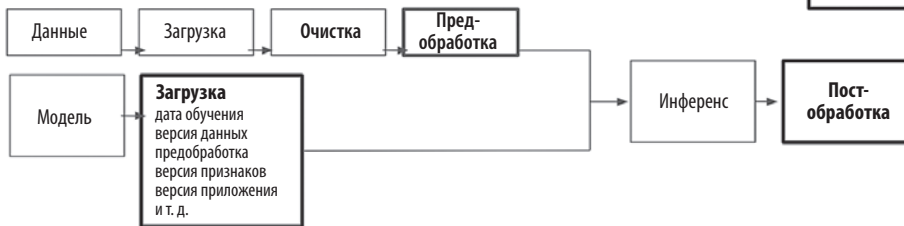


Рис. 10.10. Добавляем версию модели и приложения

Развертывание и обновление моделей — непростая задача. При создании и предоставлении инфраструктуры крайне важно обеспечить воспроизводимость результатов модели, развернутой в эксплуатационном окружении. Это означает, что каждый вызов инференса должен быть привязан к модели, датасету, на котором была обучена модель, и версии пайплайна данных, используемого для предоставления модели.

Обработка данных и DAG

Чтобы получать воспроизводимые результаты, как описано выше, пайплайн обучения также должен быть воспроизводимым и детерминированным. Для заданной комбинации из датасета, схемы предобработки и модели пайплайн обучения должен всегда выдавать одну и ту же обученную модель.

Чтобы создать модель, необходимо выполнить множество последовательных преобразований, поэтому пайплайны обычно разбивают на несколько отдельных частей. Это гарантирует, что каждая часть будет выполняться успешно и последовательно.

Один из способов упростить задачу — представить процесс перехода от сырых данных к обученной модели в виде ориентированного ациклического графа (DAG — directed acyclic graph), каждый узел которого представляет отдель-

ную операцию обработки, а каждое ребро — зависимость между двумя узлами. Эта идея лежит в основе программирования потоков данных — парадигмы программирования, на которой основана популярная МО-библиотека TensorFlow.

DAG — естественный способ визуализации предобработки. На рис. 10.11 каждая стрелка представляет собой некоторую задачу, зависящую от другой задачи. Такой способ представления позволяет упростить отдельные задачи, выразив всю сложность с помощью структуры графа.

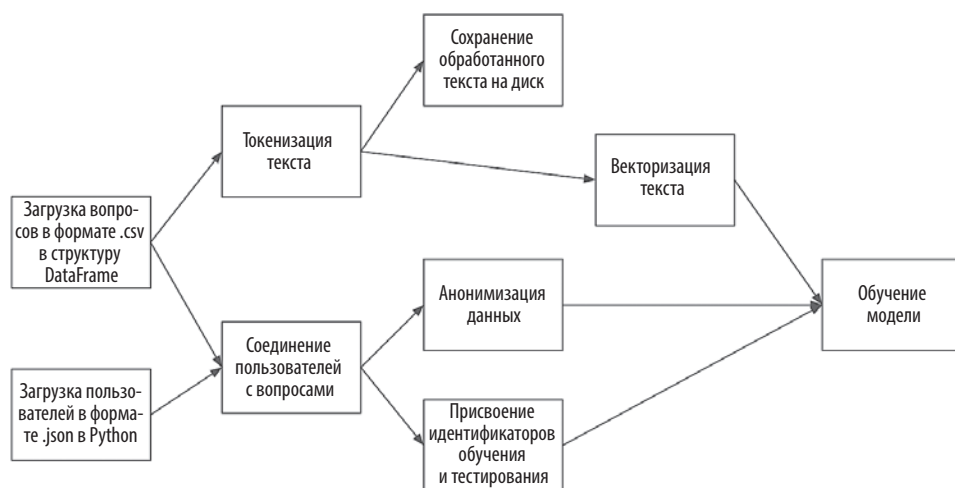


Рис. 10.11. DAG для нашего приложения

Построив такой DAG, мы можем гарантировать, что для каждой создаваемой нами модели будет выполняться одинаковый набор операций. В МО существует много инструментов для определения DAG, включая ряд активных технологий с открытым исходным кодом, таких как Apache Airflow¹ и Luigi² от компании Spotify. Оба эти пакета помогают определять DAG и предлагают дашборды, позволяющие вам отслеживать ход работы над DAG и все связанные с этим журналы.

При создании первой версии пайплайна МО использование DAG, вероятно, будет лишь осложнять задачу. Но после того как модель станет ключевой частью системы, развернутой в эксплуатационном окружении, преимущества этого подхода станут для вас вполне очевидными. После того как вы начнете регулярно дообучать и разворачивать модели, любой инструмент, способный

¹ <https://oreil.ly/8ztqj>

² <https://oreil.ly/jQFj8>

помочь вам в систематизации, отладке и версионировании пайплайна, будет серьезно экономить время.

Давайте рассмотрим в заключительной части этой главы еще один, более непосредственный способ гарантировать эффективную работу модели, который состоит в том, чтобы просто спросить пользователей.

Запрос обратной связи

В этой главе мы рассмотрели системы, способствующие тому, чтобы каждый пользователь своевременно получал точный результат. Чтобы гарантировать качество результатов, мы использовали тактики, позволяющие выявлять неточности в предсказаниях модели. Однако почему бы нам не спросить самих пользователей?

Вы можете получать обратную связь от пользователей, запрашивая ее напрямую или оценивая неявные признаки. При отображении предсказания модели можно запрашивать обратную связь, предоставляя пользователю возможность как-то оценить и скорректировать предсказание. Вы можете просто выводить диалоговое окно с вопросом о том, было ли предсказание полезным, либо применить какой-то менее прямолинейный подход.

Так, например, приложение для планирования бюджета Mint автоматически распределяет все операции по счету по категориям, например «Путешествия», «Питание» и т. д. Как показано на рис. 10.12, каждая категория отображается в пользовательском интерфейсе как поле, которое пользователь может редактировать. Такие системы позволяют получать ценную обратную связь для постоянного совершенствования моделей, делая это менее навязчиво по сравнению с прямым опросом пользователей.

Поскольку пользователи не могут дать обратную связь для каждого предсказания модели, для оценки производительности важно найти неявные признаки обратной связи. Можно понять, насколько полезны результаты модели, наблюдая за совершаемыми пользователем действиями.

При всей своей полезности неявные признаки труднее интерпретировать. Вам вряд ли удастся найти признак, всегда коррелирующий с качеством модели; он может коррелировать с качеством лишь в общей совокупности. Например, логично предположить: если пользователь рекомендательной системы кликает по рекомендуемому объекту, то рекомендация полезна. Это не всегда верно (ведь пользователи иногда нажимают не туда!), но если пользователи все же чаще кликают по нужным объектам, чем по ненужным, то это вполне подходящий неявный признак.

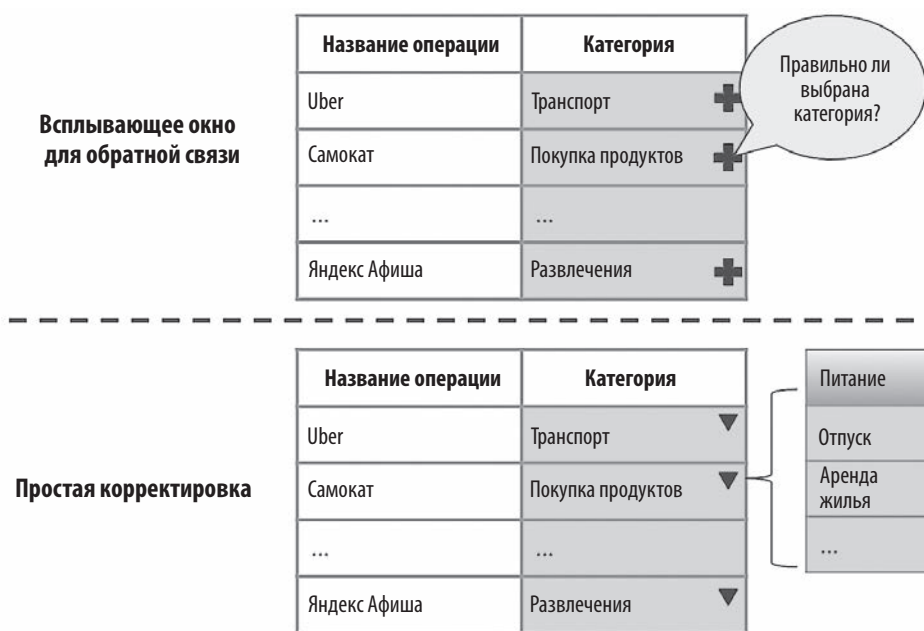


Рис. 10.12. Позвольте пользователям самим исправлять ошибки

Как показано на рис. 10.13, собрав такую информацию, вы можете оценить, насколько часто пользователи находят результаты полезными. Однако поиск неявных признаков несет за собой дополнительные риски, связанные со сбором и хранением данных, и может приводить к образованию негативных циклов обратной связи, о которых мы говорили в главе 8.

Сформировав механизмы неявной обратной связи, вы можете собирать ценную дополнительную информацию. При этом действия пользователей часто представляют собой сочетание неявной и явной обратной связи.

Допустим, что мы дополнили рекомендации нашего МО-редактора кнопкой «Задать этот вопрос в сети Stack Exchange». Посмотрев, сколько раз кликнули по этой кнопке, мы сможем определить относительное количество хороших рекомендаций, которые можно опубликовать в качестве вопросов. Добавляя эту кнопку, мы не задаем пользователю прямой вопрос о том, полезна ли для него наша рекомендация, а позволяем ему выполнить действие, которое служит для нас «слабой меткой», отражающей качество вопроса (освежить знания о слабо размеченных данных можно в разделе «Типы данных» на с. 32).

Явная и неявная обратная связь от пользователей не только позволяет получать ценные обучающие данные, но также часто является простейшим способом

выявления деградации эффективности МО-продукта. Хотя в идеале ошибки должны перехватываться до того, как они отобразятся пользователям, отслеживание такой обратной связи помогает быстрее выявлять и исправлять ошибки. В главе 11 мы поговорим об этом подробнее.

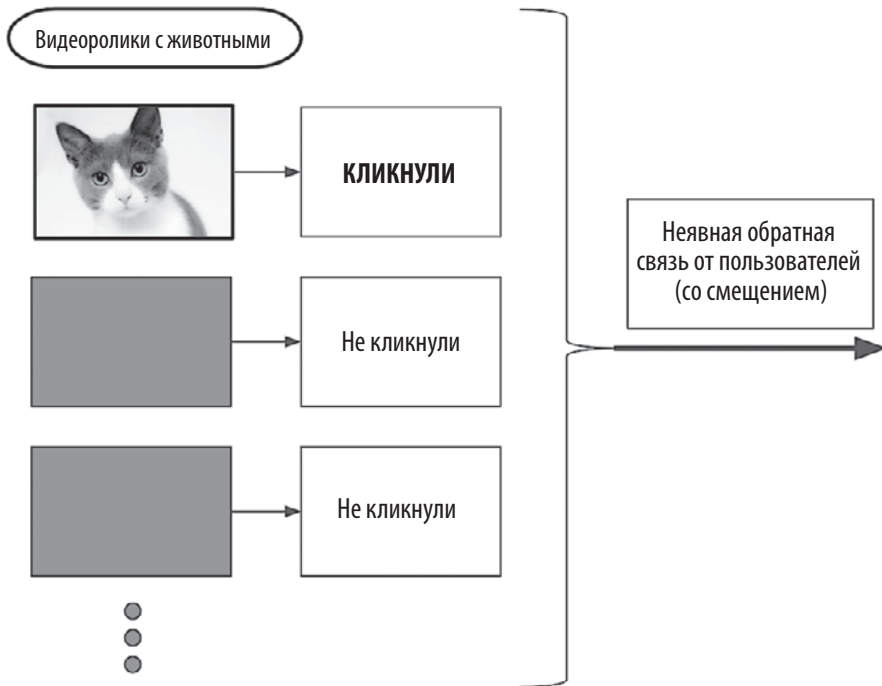


Рис. 10.13. Использование действий пользователя в качестве источника обратной связи

Стратегии развертывания и обновления моделей могут сильно варьироваться в зависимости от размера команды разработчиков и их опыта в сфере МО. Некоторые из описанных в этой главе решений будут слишком сложны для такого прототипа, как наш МО-редактор. С другой стороны, команды разработчиков, вложившие в МО значительное количество ресурсов, сумели создать сложные системы, позволяющие упростить процесс развертывания и гарантировать высокий уровень качества. В интервью ниже Крис Муди (Chris Moody), возглавляющий в компании Stitch Fix¹ подразделение инструментов на базе ИИ, расскажет о том, как подходят к развертыванию МО-моделей его коллеги.

¹ Stitch Fix — онлайн-сервис по подбору индивидуального стиля в США и Великобритании. — *Примеч. ред.*

Крис Муди: специалисты по данным отвечают за весь пайплайн моделирования

Крис Муди в прошлом занимался физикой в Калифорнийском технологическом институте и Калифорнийском университете в Санта-Крузе, а сейчас возглавляет подразделение инструментов на базе ИИ в компании Stitch Fix. Он увлекается обработкой естественного языка и имеет опыт применения глубокого обучения, вариационных методов и гауссовских процессов. Он принимал участие в создании библиотеки глубокого обучения Chainer¹ и сверхбыстрой версии алгоритма t-SNE на базе метода Барнса — Хата для библиотеки scikit-learn², а также написал одну из весьма немногочисленных Python-библиотек для факторизации разреженных тензоров³. Он также создал свою собственную модель для обработки естественного языка, которая называется lda2vec⁴.

Вопрос. За какую часть жизненного цикла модели отвечают специалисты по данным в компании Stitch Fix?

Ответ. В компании Stitch Fix специалисты по данным отвечают за весь пайплайн моделирования. Этот пайплайн включает в себя множество вещей: формирование идей, прототипирование, проектирование и отладку, ETL-процедуры и обучение моделей с использованием scikit-learn, pytorch и R. Кроме того, специалисты по данным отвечают за настройку систем для получения метрик и проверку моделей на предмет их «исправности». Наконец, они запускают A/B-тесты, отслеживают ошибки и журналы и при необходимости разворачивают обновленные версии моделей в зависимости от результатов наблюдений. Выполнение всего перечисленного возможно благодаря результатам работы команды по разработке платформы и проектированию.

Вопрос. Каким образом разработчики платформы упрощают выполнение задач, стоящих перед специалистами по данным?

Ответ. Задача разработчиков платформы — правильно подобрать абстракции для моделирования. Это означает, что они должны понимать, как работает специалист по данным. Разработчики создают не отдельный пайплайн для специалистов по данным, работающих над тем или иным проектом, а решения, которые позволяют им делать это самостоятельно, то есть в общем случае они создают инструменты, позволяющие специалистам по данным держать под контролем весь рабочий процесс. Это позволяет разработчикам направлять больше усилий на улучшение платформы и меньше — на создание «одноразовых» решений.

¹ <http://chainer.org/>

² <https://oreil.ly/t3Q0k>

³ https://oreil.ly/tS_qD

⁴ <https://oreil.ly/t7XFr>

Вопрос. Как вы оцениваете производительность моделей после их развертывания?

Ответ. Сильной стороной компании Stitch Fix является то, что люди и алгоритмы здесь работают вместе. Например, Stitch Fix направляет много усилий на выбор правильного способа представления информации для стилистов. В принципе, если с одной стороны у вас API, который дает доступ к вашей модели, а с другой — пользователь вроде стилиста или покупателя товаров, то как вам организовать взаимодействие между ними?

На первый взгляд, заманчивый подход — создать фронтенд, который будет просто отображать пользователям результаты алгоритма. К сожалению, при этом у пользователей будет создаваться впечатление, что у них нет контроля над алгоритмом и системой в целом, и это приведет к разочарованию в случае плохой работы системы. Вместо этого вы должны рассматривать взаимодействие как цикл обратной связи, позволяя пользователям исправлять и корректировать результаты. При этом пользователи обучают алгоритмы и оказывают гораздо большее влияние на процесс в целом. Кроме того, такой подход позволяет собирать помеченные данные для оценки производительности моделей.

Чтобы сделать это хорошо, специалисты по данным должны подумать, как предоставить доступ к модели, чтобы одновременно и максимально упростить стоящую перед пользователем задачу, и позволить ему улучшить модель. Поскольку специалисты по данным лучше других знают, какая обратная связь будет наиболее полезна для их моделей, то они должны иметь контроль над всем процессом вплоть до этого момента. Это позволяет им перехватывать любые ошибки, поскольку они видят весь цикл обратной связи.

Вопрос. Как вы отслеживаете состояние моделей и производите их отладку?

Ответ. Когда разработчики создают прекрасные инструменты, мониторинг и отладка становятся намного проще. Компания Stitch Fix создала внутренний инструмент, который принимает пайплайн моделирования и создает Docker-контейнер, проверяет аргументы и возвращаемые типы, предоставляет пайплайн инференса в виде API, разворачивает его в нашей инфраструктуре и создает соответствующий дашборд. Этот инструмент позволяет специалистам по данным напрямую исправлять любые ошибки, возникающие во время или после развертывания. Мы также обнаружили, что такой подход, при котором специалисты по данным отвечают за поиск и устранение ошибок в моделях, подталкивает их к использованию простых и надежных моделей, которые не так часто дают сбой. Контролируя весь пайплайн, они стремятся сделать его как можно более эффективным и надежным, вместо того чтобы использовать предельно сложную модель.

Вопрос. Как вы разворачиваете новые версии моделей?

Ответ. Специалисты по данным проводят эксперименты, используя специально разработанный сервис для A/B-тестирования, который позволяет производить

тонкую настройку параметров. После этого они анализируют результаты тестирования и, если по мнению команды эти результаты выглядят убедительно, самостоятельно разворачивают новую версию.

Что касается развертывания, то мы используем «канареечный» подход, при котором сначала развертываем новую версию на один экземпляр, а затем постепенно обновляем другие экземпляры при одновременном отслеживании производительности. При этом специалисты по данным имеют доступ к дашборду, который по мере развертывания отображает непрерывно отслеживаемые показатели производительности и количество экземпляров старой и новой версии.

Заключение

В этой главе мы рассмотрели методы повышения устойчивости ответов за счет проактивного выявления возможных ошибок модели и поиска способов их устранения. Это включает в себя детерминированные стратегии проверки и использование фильтрующих моделей. Мы также рассмотрели некоторые проблемы, связанные с поддержанием эксплуатируемой модели в актуальном состоянии. Затем обсудили способы оценки производительности модели. Наконец, ознакомились с практическим опытом компании, которой приходится часто проводить крупномасштабное развертывание МО-продуктов.

В главе 11 мы рассмотрим ряд дополнительных методов для отслеживания производительности моделей и оценки состояния МО-приложения с помощью различных показателей.

Мониторинг и обновление моделей

После развертывания модели вы должны отслеживать ее производительность, как и в случае любого ПО. Аналогично тому, что говорилось о тестировании в разделе «Тестирование МО-кода» на с. 167, для мониторинга применимы стандартные рекомендации. И аналогично вам потребуется учесть некоторые дополнительные, специфичные для МО-моделей моменты.

В этой главе мы поговорим о мониторинге МО-моделей. А если точнее, мы получим ответ на следующие три вопроса:

1. Зачем мониторить модели?
2. Как мониторить модели?
3. Какие действия должен направлять проводимый нами мониторинг?

Сначала давайте поговорим о том, каким образом мониторинг моделей поможет решить, когда развернуть новую версию или начать искать имеющиеся в эксплуатации окружении проблемы.

Мониторинг спасает жизни

Цель мониторинга — следить за здоровьем системы. В случае МО-моделей это означает отслеживание их производительности и качества предсказаний.

Если изменение привычек пользователей ведет к выдаче моделью некачественных результатов, хорошая система мониторинга позволит вам очень быстро это заметить и принять ответные меры. Теперь давайте поговорим о том, какие основные проблемы сумеет перехватить мониторинг.

Мониторинг для определения частоты обновлений

Как мы узнали в разделе «Актуальность данных и сдвиг распределения» на с. 50, чтобы поддерживать заданный уровень производительности, большинство моделей надо регулярно обновлять. При этом можно мониторить момент, когда модель перестает быть актуальной и возникает необходимость в ее дообучении.

Допустим, что мы используем получаемую от пользователей неявную обратную связь (например, в виде кликов по рекомендациям) для оценки точности модели. Непрерывно отслеживая точность модели, мы можем определить момент, когда точность опустилась ниже определенного порога, и обучить новую модель. На рис. 11.1 показана хронология этого процесса — дообучение будет проводиться при каждом падении точности ниже заданного порога.

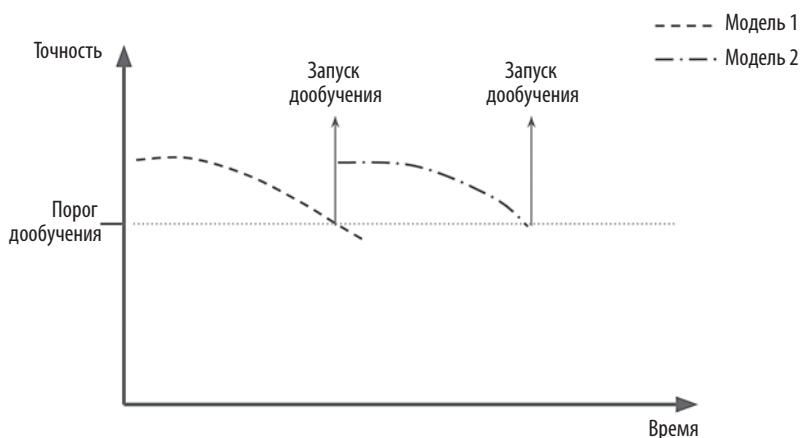


Рис. 11.1. Мониторинг для выбора момента запуска развертывания новой модели

Перед тем как приступить к развертыванию обновленной модели, надо убедиться в том, что новая модель работает лучше предыдущей. Как это сделать, мы расскажем далее в разделе «CI/CD для МО» на с. 261. Но прежде давайте рассмотрим другие отслеживаемые аспекты, например потенциальное злоупотребление.

Мониторьте, чтобы выявить злоупотребления

В ряде случаев, например при создании системы для предотвращения попыток взлома или обнаружения мошенничества, некоторая часть пользователей активно стремится нарушить работу моделей. В таких случаях мониторинг становится ключевым способом выявления атак и оценки степени их успешности.

Система мониторинга может выявлять атаки путем обнаружения аномалий. Так, при отслеживании попыток авторизации на онлайн-портале банка система мониторинга может выдавать предупреждение в случае внезапного десятикратного увеличения количества попыток авторизации, поскольку это может быть признаком атаки.

Как показано на рис. 11.2, такая система мониторинга может выдавать предупреждение при превышении некоторого порогового значения или изменении более специфичных показателей, таких как скорость увеличения попыток авторизации. В зависимости от сложности атак рекомендуется создать модель, способную выявлять аномалии, полагаясь не только на пороговое значение.

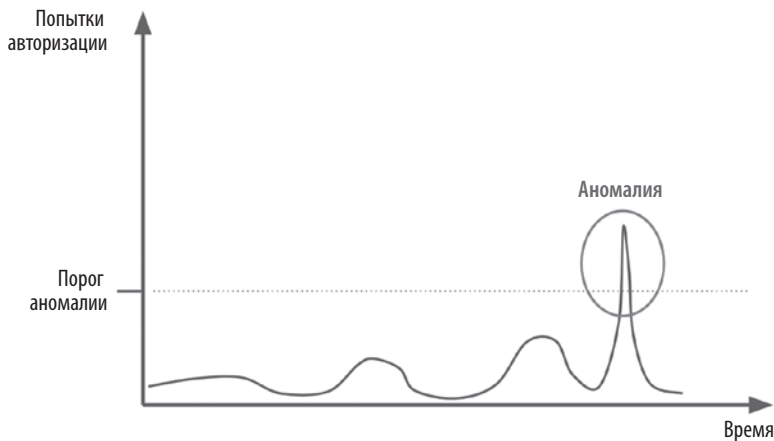


Рис. 11.2. Очевидная аномалия на дашборде мониторинга. Вы можете создать дополнительную модель для автоматического поиска таких аномалий.

Таким образом, мы можем отслеживать актуальность моделей и выявлять аномалии, однако какие еще показатели мы можем отслеживать?

Что мониторить

В программных приложениях часто отслеживаются такие показатели, как средняя продолжительность обработки запроса, относительное количество запросов, которые не удалось обработать, и количество доступных ресурсов. Такой мониторинг полезен для любого сервиса, развернутого в эксплуатационном окружении, и позволяет исправлять ситуацию до появления проблем у значительного количества пользователей.

Далее мы поговорим о том, какие еще метрики следует отслеживать, чтобы вовремя заметить снижение производительности модели.

Метрики производительности

Модель может стать устаревшей из-за изменений в распределении данных. Пример такой ситуации показан на рис. 11.3.

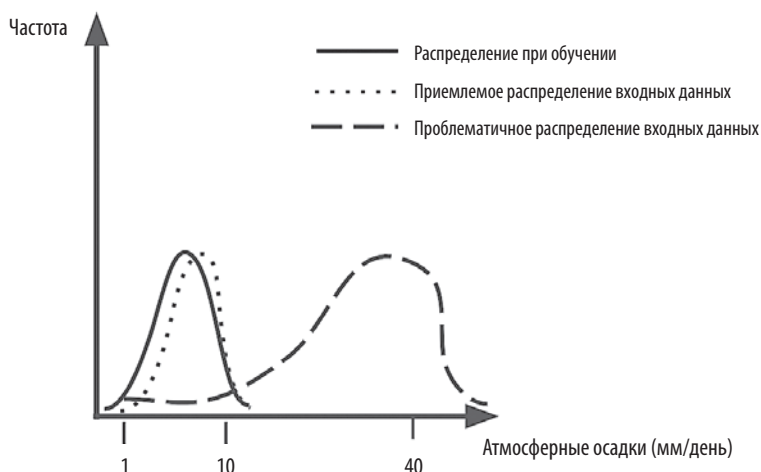


Рис. 11.3. Пример смещения распределения признаков

Что касается сдвигов, то изменяться может и распределение входных данных, и распределение предсказаний. Допустим, что наша модель пытается угадать, какой следующий фильм выберет пользователь для просмотра. При использовании в качестве входных данных одной и той же истории просмотра предсказания модели будут изменяться в зависимости от того, какие новые фильмы будут вноситься в каталог.

- *Мониторинг изменений в распределении входных данных* (что также называют «смещением признаков») дается легче, чем отслеживание распределения предсказаний, потому что получить доступ к идеальным результатам, способным удовлетворить пользователей, — непростая задача.
- *Мониторинг распределения входных данных* может сводиться к простому отслеживанию таких сводных статистических данных, как среднее значение и разброс ключевых признаков, и выдаче предупреждения в случае их отклонения относительно обучающих данных на какую-то пороговую величину.

- *Мониторинг сдвигов распределения* является более сложной задачей. Простейший подход — отслеживать распределение предсказаний модели. Как и в случае входных данных, значительные изменения в распределении предсказаний могут быть признаком снижения производительности модели. Однако распределение подходящих для пользователей результатов сложно оценить.

Бывает, что *определить* реальное положение дел трудно из-за действий модели. Чтобы лучше понять, почему так происходит, взгляните на рис. 11.4, где представлена схема работы модели по обнаружению мошенничества с кредитными картами. Слева на рисунке — распределение данных, принимаемых моделью. Для этих данных модель выдает предсказания, и на их основе выполняется код приложения, блокирующий те транзакции, которые по мнению модели являются мошенническими.

Когда транзакция блокируется, мы не можем выяснить, что произошло бы в том случае, если бы мы ее пропустили. То есть мы не можем выяснить, была ли заблокированная транзакция на самом деле мошеннической. Мы можем наблюдать и размечать только пропущенные нами транзакции. Таким образом, выполняя действия на основе предсказаний модели, мы можем наблюдать только смещенное распределение незаблокированных транзакций, показанное справа на рисунке.

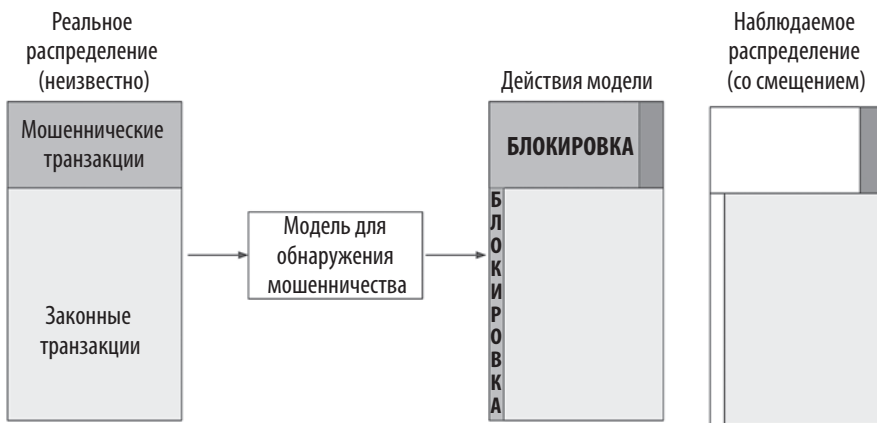


Рис. 11.4. Выполнение действий на основе предсказаний модели может привести к смещению наблюдаемого распределения данных

Имея доступ только к смещенной выборке реального распределения, мы не можем правильно оценить производительность модели. На этой идее основан метод контрфактуальной оценки, который пытается определить, что бы произо-

шло в том случае, если бы мы не задействовали модель. Для выполнения такой оценки на практике следует воздержаться от запуска модели на небольшом подмножестве образцов (см. статью Лихонга Ли (Lihong Li) и др. «Контрфактуальная оценка противодействия и оптимизация показателя кликабельности для поисковых систем» («Counterfactual Estimation and Optimization of Click Metrics for Search Engines»¹)). Воздержавшись от выполнения действий на случайно выбранном подмножестве образцов, мы сможем наблюдать несмещенное распределение мошеннических транзакций. Сравнив предсказания модели с реальными результатами для случайно выбранных данных, мы можем дать начальную оценку точности и полноты предсказаний модели.

Такой подход позволяет оценить и сравнить модели, но за это приходится платить тем, что мы пропускаем некоторую часть мошеннических транзакций. Во многих случаях это будет вполне допустимый компромисс. С другой стороны, данный подход не стоит использовать, когда выдача случайных предсказаний неприемлема, как, например, в случае медицинских приложений.

В разделе «CI/CD для МО» на с. 261 мы рассмотрим ряд других подходов к сравнению моделей и принятию решения о том, какие из них следует развивать, но прежде давайте обсудим, какие еще ключевые типы показателей необходимо мониторить.

Бизнес-метрики

Как мы видели на протяжении всей книги, наибольшая важность — у показателей, связанных с продуктом и бизнес-целями. Они могут служить мерилем того, насколько эффективно работает наша модель. Если пользователи не будут кликать по результатам поисковой выдачи, это означает провал продукта даже при нормальном уровне всех других метрик и хорошем функционировании остальной части развернутой системы.

Поэтому необходимо тщательно отслеживать метрики продукта. Для поисковых и рекомендательных систем такой мониторинг может сводиться к отслеживанию кликабельности (CTR), то есть отношения числа пользователей, кликнувших по рекомендации, к общему числу увидевших рекомендацию.

В некоторых случаях полезно модифицировать продукт так, чтобы было проще отслеживать его успешность, подобно тому, как мы показывали это в разделе «Запрос обратной связи» на с. 249. Если вы помните, там предлагалось добавить кнопку для публикации вопроса, однако мы могли бы мониторить и более подробную обратную связь. Если для применения рекомендации пользователю нужно кликнуть по ней, то мы будем отслеживать, какие рекомендации реально применялись, и использовать эти данные для обучения новой версии модели.

¹ <https://arxiv.org/abs/1403.1891>

На рис. 11.5 показана разница между обобщенным подходом (слева) и детализированным (справа).

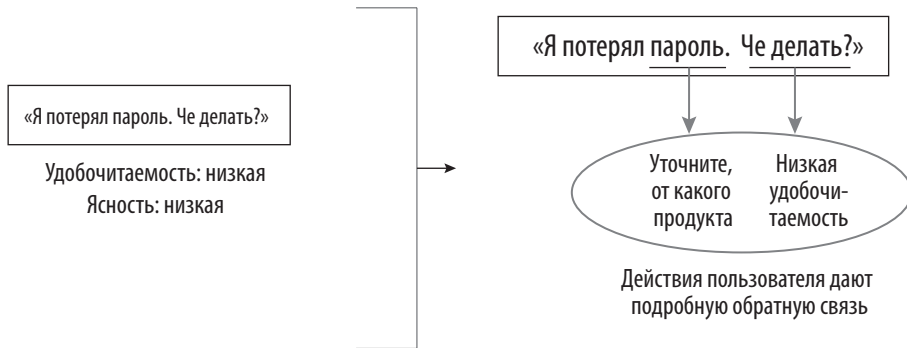


Рис. 11.5. Рекомендации на уровне слов увеличивают возможность получения обратной связи от пользователей

Наш прототип МО-редактора вряд ли будет использоваться так часто, чтобы с помощью описанного метода получить большой датасет. Однако если бы речь шла о создании продукта, нуждающегося в дальнейшей поддержке, то сбор такой обратной связи позволил бы нам получать точную информацию о том, какие рекомендации пользователи находят наиболее полезными.

Теперь, когда мы выяснили, зачем и как мониторить модели, давайте посмотрим, как устранять выявленные мониторингом проблемы.

CI/CD для МО

Под сокращением «CI/CD» понимаются «непрерывная интеграция» (continuous integration, CI) и «непрерывная поставка» (continuous delivery, CD). Если не вдаваться в подробности, то непрерывная интеграция (CI) — это процесс, позволяющий нескольким разработчикам регулярно вносить свой код в общую кодовую базу, а непрерывная поставка (CD) фокусируется на том, чтобы обеспечить оптимальную частоту выпуска новых версий программного обеспечения. Использование методов непрерывной интеграции и поставки позволяет отдельным разработчикам и организациям быстро дорабатывать и улучшать приложение вне зависимости от того, что именно нужно сделать: дополнить продукт новыми функциями или исправить имеющиеся в нем ошибки.

Таким образом, цель CI/CD в рамках МО — упростить процесс развертывания новых моделей или обновления имеющихся. Сложность при этом заключается

не в том, как обеспечить высокую скорость выпуска обновлений, а в том, как гарантировать их качество.

Как мы видели ранее, в случае МО для гарантии, что новая модель лучше предыдущей, недостаточно использовать только некоторый набор тестов. Хороший начальный шаг — обучить новую модель и протестировать ее на резервных данных, но по большому счету ничто не позволяет определить качество модели лучше, чем ее реальная производительность.

Перед развертыванием модели для пользователей команды разработчиков часто используют способ, который в статье Шелтера (Schelter) и др. «О проблемах управления моделями машинного обучения» («On Challenges in Machine Learning Model Management»¹) называется развертыванием *в тене-вом режиме*. Этот способ предполагает, что новая модель развертывается параллельно предыдущей. На этапе инференса вычисляются и сохраняются предсказания обеих моделей, но приложение использует только предсказания предыдущей модели.

Логируя таким образом значения новых предсказаний и сравнивая их одновременно и с результатами старой версии модели, и, по возможности, с реальными значениями, разработчики могут оценить производительность новой модели в реальных условиях, не внося изменения в пользовательский опыт. Такой подход также позволяет проверить инфраструктуру, необходимую для запуска инференса с новой моделью, которая может выглядеть сложнее, чем в случае предыдущей модели. Единственное, что не получится сделать в теновом режиме, так это оценить реакцию пользователей на новую модель. Это можно сделать, лишь проведя реальное развертывание.

После того как вы протестируете модель, она будет готова к развертыванию. Развертывание новой модели влечет за собой риск того, что пользователи столкнутся с ухудшением производительности. Сведение этого риска к минимуму требует определенных усилий и должно быть в центре внимания при проведении экспериментов.

На рис. 11.6 схематично показан каждый из трех описанных подходов: от самого безопасного (оценка на тестовом наборе) до наиболее информативного и наиболее рискованного (реальное развертывание модели в эксплуатационном окружении). Обратите внимание: несмотря на то что теновый режим требует дополнительных усилий для запуска сразу двух моделей при выполнении каждого инференса, он позволяет оценить модели почти столь же безопасно, как при использовании тестового набора, и обеспечивает почти столько же информации, как при развертывании модели в эксплуатационном окружении.

¹ <https://oreil.ly/zbBjq>



Рис. 11.6. Способы оценки модели: от самого безопасного и наименее точного до самого рискованного и наиболее точного

Поскольку разворачивать модели в эксплуатационном окружении может быть рискованно, команды разработчиков используют методы постепенного внедрения изменений, сначала отображая новые результаты лишь некоторому подмножеству пользователей. Давайте поговорим о том, как это делается.

А/В-тестирование и эксперименты

Цель экспериментов в МО — максимизировать шансы на использование лучшей модели и минимизировать затраты на тестирование неэффективных моделей. Существует много подходов к проведению экспериментов, и самым популярным из них является А/В-тестирование.

В основе А/В-тестирования лежит простой принцип: выборочной части пользователей вы предоставляете новую модель, а остальным — прежнюю. Обычно при этом выбирается более крупная контрольная группа для текущей модели и менее крупная экспериментальная группа, на которой проверяется новая версия модели. Проводя этот эксперимент в течение длительного времени, вы сможете сравнить результаты обеих групп и выбрать более эффективную модель.

Как видно из рис. 11.7, сначала вы случайным образом отбираете пользователей из общей совокупности и относите их к тестовому набору. Затем на этапе инференса каждому пользователю предлагается та или иная модель в зависимости от того, к какой группе он отнесен.

Хотя в основе А/В-тестирования лежит простая идея, при его использовании приходится решать ряд сложных вопросов, касающихся организации экспериментов: как отобрать контрольную и экспериментальную группы, какой должна быть продолжительность теста, как выбрать более эффективную модель и т. д.

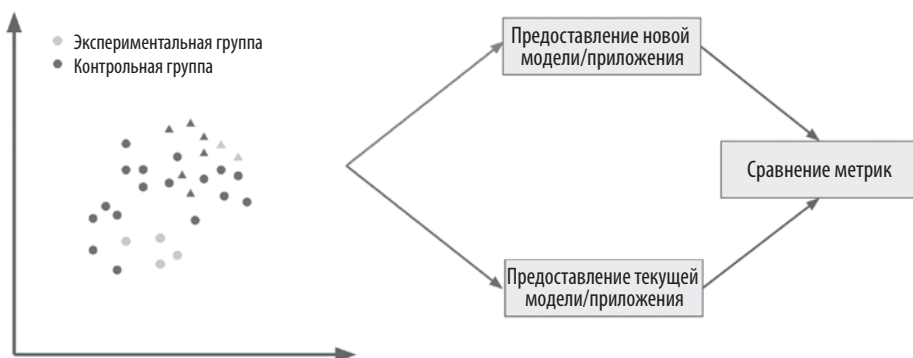


Рис. 11.7. Пример А/В-теста

Кроме того, А/В-тестирование требует создания дополнительной инфраструктуры, позволяющей предоставлять разным пользователям разные модели. Давайте подробнее остановимся на каждой из этих проблем.

Отбор групп и продолжительность теста

Для А/В-тестирования необходимо выполнить несколько условий. Во-первых, пользователи в обеих группах должны иметь как можно меньше различий, чтобы любые наблюдаемые расхождения в результатах объяснялись исключительно использованием разных моделей, а не различиями в когортах. Результаты экспериментов нельзя будет назвать убедительными, если в группу А будут входить только активные пользователи, а в группу В — только эпизодические.

Во-вторых, чтобы сделать статистически значимое заключение, экспериментальная группа В должна быть как можно меньше, поскольку вы должны ограничить доступ к потенциально менее производительной модели. На подобный компромисс приходится идти и при выборе продолжительности теста: при слишком малой продолжительности вы можете получить слишком мало информации, а при слишком большой рискуете потерять своих пользователей.

Учитывать эти два ограничения и так достаточно сложно, однако подумайте о том, насколько усложняется ситуация для крупной компании, где сотни специалистов по обработке данных одновременно выполняют десятки А/В-тестов. Несколько разных А/В-тестов могут одновременно тестировать один

и тот же аспект пайплайна, поэтому трудно точно оценить влияние того или иного теста. Когда компании выходят на такой масштаб, им приходится создавать целые платформы для проведения экспериментов. Прочтите описание фреймворка ERF, используемого в компании Airbnb, из статьи Джонатана Паркса (Jonathan Parks) «Масштабирование платформы для проведения экспериментов в компании Airbnb» («Scaling Airbnb's Experimentation Platform»¹); описание платформы XP, используемой в компании Uber, представленное в статье А. Деба (A. Deb) и др. «Внутреннее устройство платформы для проведения экспериментов, используемой в компании Uber» («Under the Hood of Uber's Experimentation Platform»); или описание проекта с открытым исходным кодом Wasabi, запущенного компанией Intuit, представленное в соответствующем GitHub-репозитории².

Выбор более эффективного варианта

Большинство А/В-тестов проводит сравнение разных групп по некоторому показателю, например кликабельности (CTR). К сожалению, для выбора более эффективной версии недостаточно просто посмотреть, для какой группы был получен более высокий CTR.

Значения любого показателя могут колебаться в силу естественных причин, поэтому сначала нужно определить, являются ли полученные результаты статистически значимыми. И поскольку мы оцениваем разницу между двумя группами пользователей, для этого чаще всего используются двухвыборочные тесты проверки гипотез.

Чтобы ваш эксперимент был убедительным, его следует проводить с достаточным количеством данных. Каким должно быть это количество, зависит от величины оцениваемой переменной и масштаба выявляемых изменений. На практике это можно сделать с помощью калькулятора размера выборки, предлагаемого Эваном Миллером (Evan Miller)³.

Перед запуском эксперимента также важно определиться с его продолжительностью и размером каждой группы. Если вместо этого вы будете постоянно оценивать значимость непрерывно выполняемого А/В-теста и посчитаете его успешно завершенным после того, как увидите значимый результат, вы получите ошибку множественного тестирования гипотез. Суть такой ошибки состоит в переоценке значимости эксперимента за счет того, что вы находите статистическую значимость результата там, где ее нет (хорошее разъяснение этой проблемы, опять же, дает в своей статье Эван Миллер⁴).

¹ <https://oreil.ly/VFcxu>

² <https://oreil.ly/txQJ2>

³ <https://oreil.ly/g4Bs3>

⁴ <https://oreil.ly/YbhmU>



Хотя большинство экспериментов фокусируется на сравнении величин какой-то одной метрики, важно учитывать и другие факторы. Если одновременно с увеличением средней величины CTR в два раза увеличивается количество пользователей, прекративших использование продукта, то такую модель, вероятно, нельзя считать более удачной.

Кроме того, при проведении А/В-тестов следует учитывать результаты, получаемые для разных сегментов пользователей. Если одновременно с увеличением средней величины CTR происходит резкое падение этого показателя на некотором сегменте пользователей, то вам вряд ли стоит разворачивать новую модель.

Для успешной реализации эксперимента вы должны отнести каждого пользователя к той или иной группе, отследить это распределение и получить на его основе разные результаты. Это требует создания дополнительной инфраструктуры, что мы и обсудим далее.

Создание инфраструктуры

Проведение экспериментов также предъявляет определенные требования к инфраструктуре. Простейший метод запуска А/В-теста — сохранить информацию о группе, к которой принадлежит пользователь, вместе с остальной информацией о пользователе (например, в базе данных).

После этого приложение может задействовать условную логику, которая будет запускать ту или иную модель в зависимости от значения определенного поля. Этот простой метод хорошо подходит для систем с авторизованными пользователями, но его сложно использовать на неавторизованных.

Это объясняется тем, что при проведении экспериментов обычно берут независимые группы пользователей, каждой из которых предоставляется только один вариант модели. Если у нас неавторизованные пользователи, сложно гарантировать, что каждому будет всегда предоставляться один и тот же вариант модели. Если большинство пользователей увидит несколько вариантов модели, это сделает результаты эксперимента несостоятельными.

Для идентификации пользователей можно использовать и другие способы, например cookie-файлы браузера или IP-адреса. Однако это, опять же, требует создания соответствующей инфраструктуры, что часто затруднительно для небольших команд разработчиков с ограниченными ресурсами.

Другие подходы

А/В-тестирование получило широкое распространение, однако существуют и другие подходы, которые пытаются обойти некоторые из его ограничений.

Одним из таких более гибких подходов является использование «многоруких бандитов», которые позволяют непрерывно тестировать сразу множество вариантов. При этом выбор модели происходит динамически, в зависимости от эффективности каждого варианта. Схема работы «многоруких бандитов» показана на рис. 11.8. Они непрерывно отслеживают производительность каждой модели, учитывая, насколько успешно выполняется каждый направляемый ими запрос. Большинство запросов просто направляется к наилучшему на данный момент варианту, как показано слева. Некоторое небольшое подмножество запросов направляется к случайно выбранному варианту модели, как показано справа. Такой подход позволяет обновлять оценку того, какая модель лучше, и выявлять случаи, когда новая модель начинает работать более эффективно.

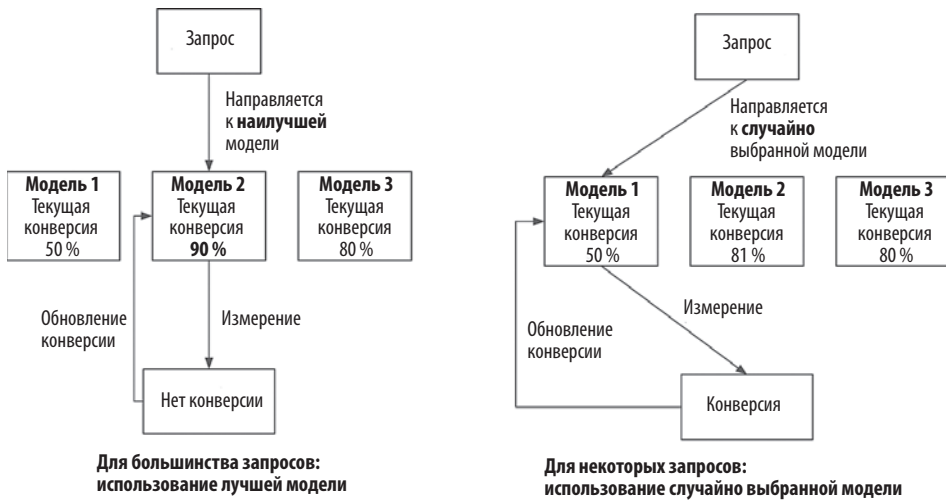


Рис. 11.8. Схема использования «многоруких бандитов»

Контекстные «многорукие бандиты» делают еще один шаг вперед. Они могут определить, какая модель является наилучшим вариантом для каждого конкретного пользователя. Для более подробного ознакомления с этим подходом я рекомендую вам прочитать краткий обзор от команды разработчиков компании Stitch Fix¹.



В этом разделе мы говорили об экспериментах для проверки моделей. Но сегодня компании начинают все чаще использовать методы экспериментирования для проверки любых значительных изменений, вносимых в приложения. Это позволяет непрерывно оценивать, какую функциональность пользователи находят полезной и насколько хорошо работают новые функции.

¹ <https://oreil.ly/K5Jpx>

Учитывая, насколько сложным и уязвимым может быть проведение экспериментов, многие стартапы начали предлагать «сервисы оптимизации», позволяющие клиентам интегрировать свои приложения с размещенной в интернете платформой для экспериментов, чтобы выяснить, какой из имеющихся вариантов работает наиболее эффективно. Когда у компании нет специального подразделения, отвечающего за проведение экспериментов, использование таких решений является самым простым способом тестирования новых версий.

Заключение

Развертывание и мониторинг моделей пока остаются сравнительно новыми практиками. Они играют принципиально важную роль, позволяя вам проверять, насколько полезна та или иная модель, но в то же время требуют значительных усилий по созданию инфраструктуры и тщательному проектированию продукта.

С течением времени стали появляться платформы для экспериментов наподобие Optimizely¹, призванные в какой-то мере упростить эту работу. В идеале такие инструменты должны позволять разработчикам МО-приложений непрерывно повышать качество результатов, предоставляемых каждому пользователю.

Если мы вспомним, какие системы были рассмотрены в книге, то поймем, что лишь небольшая их часть служит для обучения моделей. Основные усилия по созданию МО-продукта приходится на работу с данными и проектирование. Однако большинство специалистов по обработке данных, работу которых мне приходилось курировать, располагали в основном лишь информацией о методах моделирования и потому чувствовали себя плохо подготовленными к выполнению всей остальной работы. Наша книга призвана восполнить этот пробел в знаниях.

Создание МО-приложения требует широкого набора навыков в различных областях: статистике, разработке программного обеспечения, продакт-менеджменте. Каждый этап процесса достаточно сложен, и каждому можно было бы посвятить отдельную книгу. Цель нашей книги — показать широкий набор инструментов, облегчающих создание МО-приложений, и помочь вам определиться, какие темы следует изучить подробнее. В частности, вы можете обратиться к рекомендуемым источникам, которые перечислены в разделе «Дополнительные ресурсы» на с. 11.

Я надеюсь, что эта книга предоставила инструменты, позволяющие вам более уверенно решать большинство задач по созданию продуктов на базе МО. Мы рассмотрели все этапы жизненного цикла МО-продукта, включая выбор мето-

¹ <https://www.optimizely.com/>

да МО на основе цели продукта, поиск и отбор данных, итеративную доработку моделей и, наконец, оценку производительности моделей и их развертывание.

Вне зависимости от того, прочитали ли вы эту книгу от корки до корки или ознакомились лишь с теми разделами, которые имеют отношение к вашей работе, теперь вы должны располагать всеми необходимыми знаниями, чтобы приступить к созданию собственных приложений на базе МО. Если книга помогла вам в создании интересных приложений либо если у вас появились вопросы или замечания по ее содержанию, вы можете связаться со мной по адресу электронной почты mlpoweredapplications@gmail.com. Я буду всегда рад ответить и ознакомиться с результатами вашей работы в области МО.

Об авторе

Эммануэль Амейзен уже в течение многих лет создает продукты на базе МО. В настоящее время он работает инженером по машинному обучению в компании Stripe¹. До этого он возглавлял отдел ИИ в компании Insight Data Science, где курировал более 150 проектов МО. А еще раньше Эммануэль работал специалистом по обработке данных в компании Zipcar², где участвовал в разработке фреймворков и сервисов, облегчающих развертывание МО-моделей в эксплуатационном окружении. Он получил образование в области МО и бизнеса, имеет диплом магистра в области ИИ от Университета Париж-Юг, магистра технических наук от Университета Париж-Сакле и магистра в области менеджмента от Парижской высшей школы коммерции.

¹ <https://www.stripe.com>

² <https://www.zipcar.com/>

Иллюстрация на обложке

Обложку книги украшает изображение широко известной разновидности бабочек, носящей название ленточник тополевый (*Limenitis populi*). Эта довольно крупная бабочка, к сожалению, все реже встречается в зоне своего обитания — на севере Африки и Азии, на Ближнем Востоке и в Европе.

Размах крыльев тополевого ленточника составляет около 7,5 сантиметра. Окраска верхней стороны крыльев темно-коричневая с белыми пятнами и темно-серыми и оранжевыми перевязями; нижняя сторона крыльев оранжевая.

Бабочки этого вида откладывают яйца в конце августа на листьях деревьев из рода тополей (главным образом на листьях тополя осинообразного, *Populus Tremula*), поскольку их гусеницы питаются только такими листьями. Гусеницы имеют на голове роговидные отростки, по мере роста они окрашиваются в маскирующие оттенки зеленого и коричневого. Гусеницы начинают расти в конце лета и к концу осени прядут защитный кокон из шелковых нитей, в котором проводят зиму. Весной, при появлении первых почек на тополях, они вылезают из кокона и сразу же начинают набирать массу для своей окончательной трансформации: в конце мая гусеница прикрепляет себя шелковыми нитями к листу и окукливается, а потом выходит из куколки уже в виде бабочки в июне-июле.

В отличие от большинства других видов бабочек, тополевые ленточники не питаются нектаром цветов, а, используя хоботок, извлекают питательные вещества, соли и микроэлементы из падали, помета животных, древесного сока и влажной почвы (иногда даже из капель человеческого пота). Этих бабочек, видимо, привлекает запах разлагающейся органики.

В последние годы численность тополевых ленточников снижается по мере того, как все большие площади лиственных лесов вырубаются под застройку; однако, согласно классификации Красной книги Международного союза охраны природы, эти бабочки относятся к числу видов, «вызывающих наименьшие опасения». Многие из тех видов животных, которые изображены на обложках книг от издательства O'Reilly, находятся под угрозой исчезновения; все они являются важной частью нашего мира.

Представленную на обложке цветную иллюстрацию создал Хосе Марзан-младший (Jose Marzan Jr.) на основе черно-белой гравюры, взятой из *Энциклопедического словаря Мейера* (Meyers Kleines Lexicon).

Эммануэль Амейзен

**Создание приложений машинного обучения:
от идеи к продукту**

Перевел с английского С. Черников

Руководитель дивизиона	<i>Ю. Сергиенко</i>
Ведущий редактор	<i>Е. Строганова</i>
Литературные редакторы	<i>А. Алимова, Ю. Зорина</i>
Художественный редактор	<i>В. Мостипан</i>
Корректор	<i>Н. Викторова, М. Лауконен</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: 10.2022. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные
профессиональные, технические и научные.

Импортёр в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 12.08.22. Формат 70×100/16. Бумага офсетная. Усл. п. л. 20,640. Тираж 500. Заказ 0000.