

Андрей Бурков

Инженерия машинного обучения

Андрей Бурков

Инженерия машинного обучения

Machine Learning Engineering

Andriy Burkov

Инженерия машинного обучения

Андрей Бурков



Москва, 2022

УДК 004.4
ББК 32.972
Б91

Бурков А.

Б91 Инженерия машинного обучения / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2022. – 306 с.: ил.

ISBN 978-5-93700-125-2

Книга представляет собой подробный обзор передовых практик и паттернов проектирования в области прикладного машинного обучения. В отличие от многих учебников, уделяется внимание инженерным аспектам МО. Рассматриваются сбор, хранение и предобработка данных, конструирование признаков, а также тестирование и отладка моделей, развертывание и вывод из эксплуатации, сопровождение на этапе выполнения и в процессе эксплуатации. Главы книги можно изучать в любом порядке.

Издание будет полезно тем, кто собирается использовать машинное обучение в крупномасштабных проектах. Предполагается, что читатель знаком с основами МО и способен построить модель при наличии подходящим образом отформатированного набора данных.

Дизайн обложки разработан с использованием ресурса [freepik.com](https://www.freepik.com)

УДК 004.4
ББК 32.972

Copyright Title of English-language edition: "Machine Learning Engineering", ISBN 978-1-9995795-7-9 published by Andriy Burkov. Copyright © 2020 Andriy Burkov. Russian-language edition copyright © 2022 by DMC Press Publishing. All rights reserved.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

ISBN 978-1-9995795-7-9 (англ.)
ISBN 978-5-93700-125-2 (рус.)

© Andriy Burkov, 2020
© Перевод, оформление, издание,
ДМК Пресс, 2022

*Теоретически между теорией и практикой
нет никакой разницы. Но на практике она есть.*
– Бенджамин Брюстер

*Идеальный план проекта –
сначала составить список всех неизвестных.*
– Билл Лэнгли

*Привлечение средств – это искусственный интеллект.
Найм на работу – машинное обучение.
Реализация – это линейная регрессия.
А отладка – это printf().*
– Бэррон Шварц

Содержание

От издательства	14
Вступительное слово	15
Предисловие	17
Глава 1. Введение	19
1.1. Обозначения и определения	19
1.1.1. Структуры данных	19
1.1.2. Заглавная сигма	21
1.2. Что такое машинное обучение	21
1.2.1. Обучение с учителем	22
1.2.2. Обучение без учителя	23
1.2.3. Обучение с частичным привлечением учителя	24
1.2.4. Обучение с подкреплением	24
1.3. Терминология машинного обучения	25
1.3.1. Данные, используемые прямо и косвенно	25
1.3.2. Первичные и аккуратные данные	26
1.3.3. Обучающие и зарезервированные наборы	27
1.3.4. Ориентир	28
1.3.5. Конвейер машинного обучения	28
1.3.6. Параметры и гиперпараметры	29
1.3.7. Классификация и регрессия	29
1.3.8. Обучение на основе модели и обучение на основе экземпляров	30
1.3.9. Поверхностное и глубокое обучение	30
1.3.10. Обучение и оценивание	30
1.4. Когда следует использовать машинное обучение	31
1.4.1. Когда задача слишком сложна для кодирования	31
1.4.2. Когда задача постоянно меняется	32
1.4.3. Когда речь идет о задаче восприятия	32
1.4.4. Когда это неизученное явление	32
1.4.5. Когда задача имеет простую целевую функцию	33
1.4.6. Когда это экономически выгодно	33
1.5. Когда не следует использовать машинное обучение	34
1.6. Что такое инженерия машинного обучения	34
1.7. Жизненный цикл проекта машинного обучения	36
1.8. Резюме	37
Глава 2. Прежде чем приступить к проекту	39
2.1. Определение приоритетов проекта машинного обучения	39
2.1.1. Последствия машинного обучения	39
2.1.2. Стоимость машинного обучения	40

2.2. Оценивание сложности проекта машинного обучения	41
2.2.1. Неизвестные	41
2.2.2. Упрощение задачи	42
2.2.3. Нелинейный прогресс	43
2.3. Определение цели проекта машинного обучения	43
2.3.1. Что модель может делать	43
2.3.2. Свойства успешной модели	44
2.4. Организация группы машинного обучения	45
2.4.1. Две традиции	45
2.4.2. Члены группы машинного обучения	46
2.5. Причины провалов проектов машинного обучения	47
2.5.1. Нехватка квалифицированных кадров	47
2.5.2. Отсутствие поддержки со стороны руководства	48
2.5.3. Отсутствующая инфраструктура данных	48
2.5.4. Трудности с разметкой данных	49
2.5.5. Разобщенные организации и отсутствие сотрудничества	49
2.5.6. Технически невыполнимые проекты	50
2.5.7. Нестыковка между техническими и коммерческими группами	50
2.6. Резюме	51

Глава 3. Сбор и подготовка данных

3.1. Вопросы к данным	53
3.1.1. Доступны ли данные?	54
3.1.2. Насколько велик объем данных?	54
3.1.3. Пригодны ли данные для использования?	56
3.1.4. Понятны ли данные?	58
3.1.5. Надежны ли данные?	58
3.2. Типичные проблемы с данными	60
3.2.1. Высокая стоимость	60
3.2.2. Плохое качество	62
3.2.3. Зашумленность	62
3.2.4. Смещение	63
Типы смещения	63
Как избежать смещения	67
3.2.5. Низкая предсказательная способность	69
3.2.6. Устаревшие примеры	70
3.2.7. Выбросы	70
3.2.8. Просачивание данных	71
3.3. Что считать хорошими данными	72
3.3.1. Хорошие данные информативны	72
3.3.2. Хорошие данные обладают хорошим покрытием	72
3.3.3. Хорошие данные отражают реальные входы	73
3.3.4. Хорошие данные несмещенные	73
3.3.5. Хорошие данные не являются результатом петли обратной связи	73
3.3.6. У хороших данных согласованные метки	74
3.3.7. Хорошие данные достаточно велики	74
3.3.8. Сводный перечень свойств хороших данных	74

3.4. Обработка данных о взаимодействии	75
3.5. Причины просачивания данных.....	75
3.5.1. Цель является функцией от признака	76
3.5.2. Признак скрывает цель.....	76
3.5.3. Признак из будущего.....	77
3.6. Разбиение данных.....	78
3.6.1. Просачивание во время разбиения.....	79
3.7. Обработка отсутствия атрибутов	80
3.7.1. Методы подстановки данных.....	80
3.7.2. Просачивание во время подстановки	82
3.8. Приращение данных.....	82
3.8.1. Приращение данных для изображений	82
3.8.2. Приращение данных для текста	84
3.9. Обработка несбалансированных данных	85
3.9.1. Выборка с избытком.....	86
3.9.2. Выборка с недостатком.....	87
3.9.3. Гибридные стратегии	87
3.10. Стратегии выборки данных.....	88
3.10.1. Простая случайная выборка	89
3.10.2. Систематическая выборка.....	90
3.10.3. Стратифицированная выборка.....	90
3.11. Хранение данных	90
3.11.1. Форматы данных	91
3.11.2. Уровни хранения данных	92
3.11.3. Версионирование данных	94
3.11.4. Документация и метаданные	96
3.11.5. Жизненный цикл данных.....	96
3.12. Дополнительные рекомендации по работе с данными	97
3.12.1. Воспроизводимость.....	97
3.12.2. Сначала данные, потом алгоритм.....	97
3.13. Резюме	98

Глава 4. Конструирование признаков	100
4.1. Зачем конструировать признаки.....	100
4.2. Как конструируются признаки.....	101
4.2.1. Конструирование признаков для текста	102
4.2.2. Почему мешок слов работает.....	105
4.2.3. Преобразование категориальных признаков в числа.....	105
4.2.4. Хеширование признаков	108
4.2.5. Тематическое моделирование	109
4.2.6. Признаки для временных рядов	112
4.2.7. Проявите свои творческие способности.....	114
4.3. Штабелирование признаков.....	115
4.3.1. Штабелирование векторов признаков	115
4.3.2. Штабелирование индивидуальных признаков	116
4.4. Свойства хороших признаков	117

4.4.1. Высокая предсказательная способность	117
4.4.2. Быстрое вычисление	117
4.4.3. Надежность	118
4.4.4. Некоррелированность	118
4.4.5. Другие свойства	118
4.5. Отбор признаков	119
4.5.1. Отрезание длинного хвоста	119
4.5.2. Boruta	120
4.5.3. L1-регуляризация	123
4.5.4. Зависящий от задачи отбор признаков	123
4.6. Синтезирование признаков	123
4.6.1. Дискретизация признаков	124
4.6.2. Синтез признаков из реляционных данных	125
4.6.3. Синтезирование признаков по данным	126
4.6.4. Синтезирование признаков по другим признакам	127
4.7. Обучение признаков на данных	128
4.7.1. Погружения слов	128
4.7.2. Погружения документов	130
4.7.3. Погружения всего, чего угодно	131
4.7.4. Выбор размерности погружения	132
4.8. Понижение размерности	132
4.8.1. Быстрое понижение размерности методом PCA	133
4.8.2. Понижение размерности с целью визуализации	133
4.9. Масштабирование признаков	133
4.9.1. Нормировка	134
4.9.2. Стандартизация	135
4.10. Просачивание данных при конструировании признаков	136
4.10.1. Возможные проблемы	136
4.10.2. Решение	136
4.11. Хранение и документирование признаков	136
4.11.1. Файл схемы	137
4.11.2. Хранилище признаков	138
4.12. Рекомендации по конструированию признаков	141
4.12.1. Генерируйте много простых признаков	141
4.12.2. Повторно используйте унаследованные системы	141
4.12.3. Используйте идентификаторы как признаки, когда это необходимо	142
4.12.4. ...но по возможности уменьшайте количество значений	142
4.12.5. Осторожнее со счетчиками	143
4.12.6. Отбирайте признаки, когда необходимо	143
4.12.7. Тщательно тестируйте код	144
4.12.8. Синхронизируйте код, модель и данные	144
4.12.9. Изолируйте код выделения признаков	144
4.12.10. Сериализуйте модель и экстрактор признаков совместно	145
4.12.11. Протоколируйте значения признаков	145
4.13. Резюме	145

Глава 5. Обучение модели с учителем (часть 1)	147
5.1. Прежде чем приступить к работе над моделью	148
5.1.1. Проверка согласованности со схемой	148
5.1.2. Определение достижимого уровня качества	148
5.1.3. Выбор метрики качества	149
5.1.4. Выбирайте правильный ориентир	149
5.1.5. Разбиение данных на три набора	151
5.1.6. Предварительные условия для обучения с учителем	152
5.2. Представление меток для машинного обучения	153
5.2.1. Многоклассовая классификация	153
5.2.2. Многозначная классификация	154
5.3. Выбор алгоритма обучения	154
5.3.1. Основные свойства алгоритма обучения	155
5.3.2. Выборочная проверка алгоритмов	156
5.4. Построение конвейера	158
5.5. Оценивание качества модели	159
5.5.1. Метрики качества для регрессии	159
5.5.2. Метрики качества для классификации	160
5.5.3. Метрики качества для ранжирования	165
5.6. Настройка гиперпараметров	168
5.6.1. Поиск на сетке	169
5.6.2. Случайный поиск	170
5.6.3. Поиск с измельчением	170
5.6.4. Другие методы	172
5.6.5. Перекрестная проверка	172
5.7. Обучение поверхностной модели	173
5.7.1. Стратегия обучения поверхностной модели	173
5.7.2. Сохранение и восстановление модели	174
5.8. Компромисс между смещением и дисперсией	175
5.8.1. Недообучение	175
5.8.2. Переобучение	176
5.8.3. Компромисс	177
5.9. Регуляризация	179
5.9.1. L1- и L2-регуляризации	179
5.9.2. Другие формы регуляризации	180
5.10. Резюме	180
Глава 6. Обучение модели с учителем (часть 2)	183
6.1. Стратегия обучения глубоких моделей	183
6.1.1. Стратегия обучения нейронной сети	184
6.1.2. Метрика качества и функция стоимости	184
6.1.3. Стратегии инициализации параметров	187
6.1.4. Алгоритмы оптимизации	187
6.1.5. Планы уменьшения скорости обучения	191
6.1.6. Регуляризация	192
6.1.7. Определение размера сети и настройка гиперпараметров	193

6.1.8. Работа с несколькими входами	195
6.1.9. Работа с несколькими выходами.....	196
6.1.10. Перенос обучения	197
6.2. Штабелирование моделей.....	199
6.2.1. Типы ансамблевого обучения	199
6.2.2. Алгоритм штабелирования моделей	200
6.2.3. Просачивание данных при штабелировании моделей.....	201
6.3. Борьба со сдвигом распределения.....	202
6.3.1. Обработка сдвига распределения	202
6.3.2. Состязательная проверка	202
6.4. Обработка несбалансированных наборов данных	203
6.4.1. Взвешивание классов	203
6.4.2. Ансамбль перераспределенных наборов данных.....	204
6.4.3. Другие методы	205
6.5. Калибровка модели.....	205
6.5.1. Хорошо откалиброванные модели.....	205
6.5.2. Методы калибровки	207
6.6. Поиск неполадок и анализ ошибок	208
6.6.1. Причины плохого поведения модели.....	208
6.6.2. Итеративное уточнение модели.....	209
6.6.3. Анализ ошибок.....	209
6.6.4. Анализ ошибок в комплексных системах	211
6.6.5. Использование расслоенных метрик.....	212
6.6.6. Исправление неправильных меток.....	212
6.6.7. Нахождение дополнительных примеров для пометки	213
6.6.8. Поиск неполадок при глубоком обучении	213
6.7. Рекомендации	215
6.7.1. Поставляйте хорошую модель.....	215
6.7.2. Доверяйте популярным реализациям с открытым исходным кодом	215
6.7.3. Оптимизируйте важную для бизнеса меру качества.....	216
6.7.4. При обновлении начинайте с нуля.....	216
6.7.5. Избегайте каскадов коррекций.....	217
6.7.6. Используйте каскадирование моделей с осторожностью	217
6.7.7. Пишите эффективный код, компилируйте и распараллеливайте	218
6.7.8. Тестируйте на старых и новых данных.....	219
6.7.9. Больше данных лучше, чем более умный алгоритм	220
6.7.10. Новые данные лучше более изощренных признаков.....	220
6.7.11. Радуйтесь крохотным достижениям	220
6.7.12. Обеспечьте воспроизводимость	220
6.8. Резюме	221
Глава 7. Оценивание модели	224
7.1. Офлайнное и онлайнное оценивания	225
7.2. А/В-тестирование	227
7.2.1. G-критерий	228
7.2.2. Z-критерий.....	231

7.2.3. Заключительные замечания и предупреждения.....	233
7.3. Многорукий бандит	233
7.4. Статистические границы качества модели	236
7.4.1. Статистический интервал для ошибки классификации	236
7.4.2. Бутстреп статистического интервала	237
7.4.3. Бутстреп интервала предсказания для регрессии	238
7.5. Оценивание адекватности тестового набора.....	239
7.5.1. Нейронное покрытие.....	239
7.5.2. Мутационное тестирование	240
7.6. Оценивание свойств модели	240
7.6.1. Робастность	241
7.6.2. Справедливость	241
7.7. Резюме	243

Глава 8. Развертывание модели

8.1. Статическое развертывание	245
8.2. Динамическое развертывание на устройстве пользователя.....	245
8.2.1. Развертывание параметров модели	246
8.2.2. Развертывание сериализованного объекта	246
8.2.3. Развертывание в браузере.....	246
8.2.4. Плюсы и минусы	246
8.3. Динамическое развертывание на сервере	247
8.3.1. Развертывание на виртуальной машине	247
8.3.2. Развертывание в контейнере.....	248
8.3.3. Бессерверное развертывание.....	250
8.3.4. Потокное развертывание модели.....	251
8.4. Стратегии развертывания	253
8.4.1. Разовое развертывание	253
8.4.2. Немое развертывание	254
8.4.3. Канареечное развертывание.....	254
8.4.4. Многорукие бандиты	255
8.5. Автоматизированное развертывание, версионирование и метаданные	255
8.5.1. Объекты, сопровождающие модель	255
8.5.2. Синхронизация версий.....	256
8.5.3. Метаданные версии модели.....	256
8.6. Рекомендации по развертыванию модели	257
8.6.1. Эффективность алгоритма	257
8.6.2. Развертывание глубоких моделей.....	260
8.6.3. Кеширование	260
8.6.4. Формат доставки модели и кода.....	261
8.6.5. Начинайте с простой модели.....	263
8.6.6. Тестируйте на посторонних	263
8.7. Резюме.....	264

Глава 9. Выполнение, мониторинг и сопровождение модели

9.1. Свойства среды выполнения модели.....	266
9.1.1. Безопасность и корректность	267

9.1.2. Простота развертывания	267
9.1.3. Гарантии правильности модели	268
9.1.4. Простота восстановления	268
9.1.5. Предотвращение расхождений между обучением и выполнением	268
9.1.6. Предотвращение скрытых петель обратной связи	269
9.2. Режимы выполнения модели	269
9.2.1. Выполнение в пакетном режиме.....	270
9.2.2. Обслуживание запроса со стороны человека	270
9.2.3. Обслуживание запроса со стороны машины.....	272
9.3. Выполнение модели на практике	273
9.3.1. Готовность к ошибкам.....	273
9.3.2. Отношение к ошибкам.....	274
9.3.3. Готовность к изменениям и отношение к ним	275
9.3.4. Готовность к особенностям человеческой природы и отношение к ним	277
Избегайте путаницы	277
Умерьте ожидания.....	277
Завоевывайте доверие	277
Не переутомляйте пользователя.....	278
Остерегайтесь фактора отторжения	278
9.4. Мониторинг модели	278
9.4.1. Что может пойти не так?.....	279
9.4.2. Что и как мониторить	280
9.4.3. Что протоколировать	282
9.4.4. Мониторинг неправомерного использования	283
9.5. Сопровождение модели	283
9.5.1. Когда обновлять	284
9.5.2. Как обновлять.....	285
9.6. Резюме	288
Глава 10. Заключение.....	290
10.1. Сухой остаток.....	290
10.2. Что еще почитать	294
10.3. Благодарности	295
Предметный указатель.....	297

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательство «ДМК Пресс» очень серьезно относится к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Вступительное слово

Хочу открыть вам один секрет: когда говорят «машинное обучение», складывается впечатление, будто речь идет только об одной дисциплине. А вот и нет! На самом деле есть два вида машинного обучения, и они различаются так же, как придумывание новых блюд и изобретение кухонных принадлежностей. То и другое занятия достойны уважения, но путать их не надо: не станете же вы поручать шеф-повару разработку кухонной плиты, а инженеру-электрику – печь хлеб!

Увы, почти все путают эти два вида машинного обучения. Поэтому неудивительно, что так много компаний терпят крах на этапе машинного обучения. Начинаящим, похоже, никто не объяснил, что все то, что преподают на курсах машинного обучения и пишут в учебниках, – это теоретическое машинное обучение: как конструировать кухонные плиты (а также микроволновки, блендеры, тостеры, чайники ... и мойку, в которой все смешивается!) с нуля, а не как придумывать новые блюда и готовить на огромную компанию. Иными словами, если ваша цель – создавать инновационные решения конкретных задач на базе МО, то вам нужно прикладное, а не теоретическое машинное обучение. Поэтому большинство книг – не для вас.

А теперь хорошая новость! Перед вами первая книга, посвященная именно прикладному машинному обучению. Да, вы ее нашли! Настоящая прикладная иголка в стоге теоретического сена. Поздравляю, любезный читатель... если, конечно, вы не искали книгу, которая помогла бы отточить навыки проектирования алгоритмов общего назначения. Но тогда я надеюсь, что автор не будет на меня в обиде за то, что я посоветую вам купить чуть ли не любую другую книгу по МО. А эта от них отличается.

В 2016 году, разрабатывая курс Google по прикладному МО «Как подружиться с машинным обучением», полюбившийся нашим инженерам и руководителям групп – а их больше десяти тысяч, – я придала ему примерно такую же структуру, как в этой книге. А все потому, что действовать в правильном порядке очень важно в прикладных дисциплинах. Попытка выполнить определенные шаги, прежде чем будут завершены другие, может иметь печальные последствия: от зря потраченного времени до феерического краха проекта. На самом деле сходство оглавления этой книги и моего курса как раз и побудило меня прочитать ее. И в полном согласии с теорией параллельной эволюции я узрела в авторе единомышленника, терзаемого по ночам отсутствием ресурсов по прикладному машинному обучению, одной из потенциально самых полезных и вместе с тем самых недопонятых отраслей инженерной практики, – терзаемого с такой силой, что он захотел как-то исправить ситуацию. Так что если вы вознамерились захлопнуть книгу, то, пожалуйста, сделайте мне одолжение – задумайтесь на минутку, почему главы следуют именно в таком порядке. Обещаю, уже это принесет полезные плоды.

Так что же будет в книге дальше? Машинное обучение можно сравнить с руководством по приготовлению еды в массовом масштабе. Поскольку вы

еще не прочитали книгу, опишу содержание книги в кулинарных терминах. Вам нужно определить, что имеет смысл приготовить и каковы цели (*принятие решений и управление продуктом*), понять, кто является поставщиками и клиентами (*знакомство с предметной областью и деловое чутье*), как обрабатывать ингредиенты в большом количестве (*инженерия и анализ данных*), как быстро оценивать много разных сочетаний ингредиентов, пригодных для создания потенциальных рецептов блюд (*инженерная разработка прототипа*), как проверить качество рецепта (*статистика*), как эффективно превратить потенциальный рецепт в миллионы порций (*организация производства*) и как гарантировать, что блюда останутся высококачественными, даже если грузовик привез тонну картошки вместо заказанного риса (*обеспечение надежности*). Эта книга – одна из немногих, где рассматриваются все этапы технологического процесса.

Теперь самое время подпустить немного дегтя в бочку меда. Это хорошая книга. Правда. Но не идеальная. Иногда автор срезает углы – как часто делают инженеры, профессионально занимающиеся машинным обучением, – хотя в целом все изложено правильно. А поскольку предмет книги быстро эволюционирует, автор и не пытается всегда быть на переднем крае. Но даже в отсутствие совершенства книгу все равно стоит прочитать. Учитывая, как мало на рынке подробных руководств по прикладному машинному обучению, ясное и последовательное введение в эту тему дорогого стоит. Я очень рада, что такая книга появилась!

Что мне в ней очень нравится, так это полнота, с которой излагается самая важная вещь, которую нужно знать о машинном обучении: ошибки возможны... и иногда болезненные ошибки. Как любят говорить мои коллеги, занимающиеся надежностью: «Надежда на лучшее – это не стратегия». Надеяться, что ошибок не будет, – худший из возможных подходов. Эта книга устроена иначе – и лучше. Она быстро заставляет расстаться с ложным чувством безопасности, которое вы могли лелеять, надеясь построить систему ИИ, более «умную», чем вы сами. (Забудьте об этом, ничего не получится.) Затем автор становится вашим заботливым проводником по пути, на котором можно надеяться самых разнообразных ошибок, и объясняет, как их предотвратить, обнаружить и исправить. В книге прекрасно подчеркнута важность мониторинга, описаны подходы к сопровождению модели, рассказано, что делать, если что-то ломается, как спланировать резервные стратегии на случай разного рода отказов и как управлять ожиданиями пользователей (есть также раздел о том, что делать, если пользователями являются машины). Эти вопросы очень важны при практическом применении машинного обучения, но в других книгах для них часто не находится места. Впрочем, эта книга не из их числа.

Если вы собираетесь использовать машинное обучение для решения крупномасштабных практических задач, могу только порадоваться, что вы наткнулись на эту книгу. Читайте с удовольствием!

Кэсси Козырьков,
главный специалист по теории принятия решений в Google,
автор курса «MakingFriends with Machine Learning»
на облачной платформе Google

Сентябрь 2020

Предисловие

За последние несколько лет машинное обучение (МО) для многих стало синонимом искусственного интеллекта. И хотя машинное обучение как научная дисциплина существует уже несколько десятков лет, в мире найдется лишь горстка организаций, в полной мере осознавших его потенциал.

Несмотря на доступность современных библиотек, пакетов и каркасов машинного обучения с открытым исходным кодом, которые поддерживаются ведущими организациями и широким сообществом ученых и программистов, большинство компаний все еще испытывают трудности с применением машинного обучения к решению практических деловых задач.

Одна из проблем – нехватка кадров. Но, даже располагая талантливыми специалистами по машинному обучению и анализу данных, в 2020 году большинство организаций все еще тратят от 31 до 90 дней на развертывание всего одной модели, а 18 % – более 90 дней, у некоторых на доведение идеи до продукта уходит даже больше года. Основные проблемы, с которыми приходится сталкиваться при освоении потенциала МО, например управление версиями модели, воспроизводимость результатов и масштабирование, имеют скорее инженерную, чем научную природу.

Существует много книг по машинному обучению – как теоретических, так и практических. Из типичного учебника вы можете узнать о разных типах машинного обучения, об основных семействах алгоритмов, о том, как они работают и как с их помощью создавать модели из данных.

В типичном учебнике меньше внимания уделяется инженерным аспектам реализации проектов машинного обучения. Такие вопросы, как сбор, хранение и предобработка данных, конструирование признаков, а также тестирование и отладка моделей, развертывание и вывод из эксплуатации, сопровождение на этапе выполнения и в процессе эксплуатации, зачастую остаются за кадром.

Эта книга призвана заполнить пробел.

НА КОГО РАССЧИТАНА ЭТА КНИГА

Я предполагаю, что читатель знаком с основами машинного обучения и способен построить модель при наличии подходящим образом отформатированного набора данных, применяя свой любимый язык программирования или библиотеку. Если вы неуверенно владеете применением алгоритмов машинного обучения к данным и не видите разницы между логистической регрессией, методом опорных векторов и случайным лесом, то я рекомендую предварительно прочитать мою книгу «The Hundred-Page Machine Learning Book»¹.

¹ Бурков А. Машинное обучение без лишних слов. СПб.: Питер, 2020.

Целевая аудитория этой книги – аналитики данных, стремящиеся стать инженерами по машинному обучению, инженеры по машинному обучению, стремящиеся привести больше порядка в свою работу, студенты, изучающие машинное обучение, а также архитекторы программных систем, которым приходится иметь дело с моделями, разработанными аналитиками данных и инженерами по машинному обучению.

КАК ИСПОЛЬЗОВАТЬ ЭТУ КНИГУ

Эта книга представляет собой подробный обзор передовых практик и паттернов проектирования в области прикладного машинного обучения. Я рекомендую читать ее с начала до конца. Но можете читать главы в любом порядке, поскольку в них рассматриваются различные аспекты жизненного цикла проекта с использованием машинного обучения и прямых зависимостей между главами нет.

Андрей Бурков

Глава 1

Введение

Хотя предполагается, что читатель этой книги знаком с основами машинного обучения, все же важно начать с определений, чтобы у нас было общее понимание используемых в книге терминов.

Ниже я повторяю некоторые определения из главы 2 книги «Машинное обучение без лишних слов», а также даю несколько новых определений. Если вы читали мою первую книгу, то некоторые части этой главы могут показаться вам знакомыми.

После этой главы мы будем одинаково понимать такие понятия, как обучение с учителем и без учителя. У нас будет общий взгляд на прямо и косвенно используемые данные, первичные и аккуратные данные, обучающие и зарезервированные данные.

Мы будем знать о том, когда следует использовать машинное обучение, а когда этого делать не стоит, а также о различных формах машинного обучения, например: на основе модели и на основе экземпляров, глубокое и поверхностное, классификация и регрессия и т. д.

Наконец, мы определим предмет инженерии машинного обучения и представим жизненный цикл проекта машинного обучения.

1.1. ОБОЗНАЧЕНИЯ И ОПРЕДЕЛЕНИЯ

Начнем с базовых математических обозначений и определим термины и понятия, к которым часто будем обращаться в этой книге.

1.1.1. Структуры данных

Скаляром¹ называется простое числовое значение, например 15 или -3.25 . Переменные или константы, принимающие скалярные значения, обозначаются курсивом, например x или a .

Вектором называется упорядоченный список скалярных значений, именуемых атрибутами. Мы обозначаем вектор полужирным шрифтом, например \mathbf{x} или \mathbf{w} . Векторы изображаются в виде направленных стрелок, а также

¹ Если термин выделен полужирным шрифтом, значит, он присутствует в алфавитном указателе в конце книги.

точек в многомерном пространстве. Иллюстрации трех двумерных векторов $\mathbf{a} = [2, 3]$, $\mathbf{b} = [-2, 5]$ и $\mathbf{c} = [1, 0]$ приведены на рис. 1.1. Атрибут вектора обозначается курсивной буквой с верхним индексом, например: $w^{(j)}$ или $x^{(j)}$. Индекс j обозначает конкретное **измерение** вектора, т. е. позицию атрибута в списке. Например, в векторе \mathbf{a} , показанном красным цветом на рис. 1.1, $a^{(1)} = 2$ и $a^{(2)} = 3$.

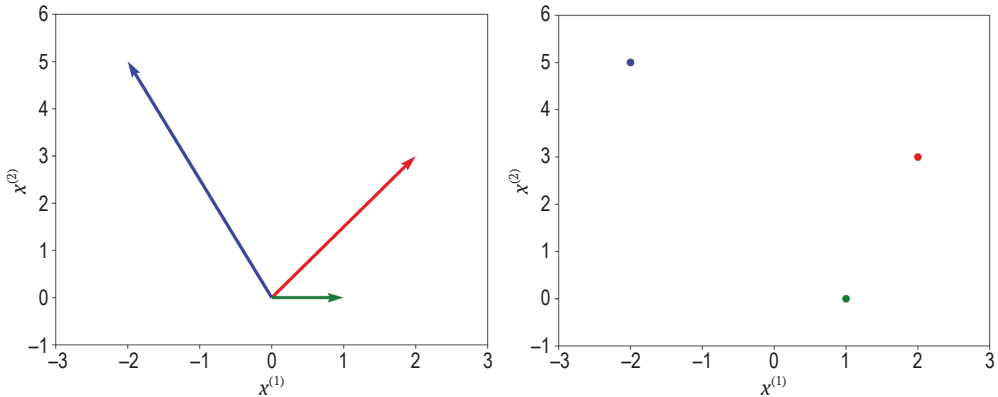


Рис. 1.1 ❖ Три вектора, представленных направленными стрелками и точками

Обозначение $x^{(j)}$ не следует путать с оператором возведения в степень, например 2 в x^2 (возведение в квадрат) или 3 в x^3 (возведение в куб). Если мы хотим применить оператор возведения в степень, скажем, в квадрат, к индексированному атрибуту вектора, то пишем: $(x^{(j)})^2$.

Переменная может иметь два и более индексов, например: $x_{i,j}^{(j)}$ или $x_{i,j}^{(k)}$. Так, в нейронных сетях $x_{i,u}^{(j)}$ обозначает j -й входной признак u -го блока в l -м слое.

Матрица – это прямоугольный массив чисел, организованный по строкам и столбцам. Ниже приведен пример матрицы с двумя строками и тремя столбцами:

$$\mathbf{A} = \begin{bmatrix} 2 & -2 & 1 \\ 3 & 5 & 0 \end{bmatrix}.$$

Матрицы обозначаются полужирными заглавными буквами, например \mathbf{A} или \mathbf{W} . Из примера матрицы \mathbf{A} выше видно, что матрицы можно трактовать как регулярные структуры, состоящие из векторов. И действительно, столбцами приведенной выше матрицы \mathbf{A} являются векторы \mathbf{a} , \mathbf{b} и \mathbf{c} , показанные на рис. 1.1.

Множеством называется неупорядоченная коллекция неповторяющихся элементов. Множество обозначается каллиграфической заглавной буквой, например \mathcal{S} . Множество чисел может быть конечным (содержать фиксированное количество значений). В этом случае его элементы перечисляются в фигурных скобках, например $\{1, 3, 18, 23, 235\}$ или $\{x_1, x_2, x_3, x_4, \dots, x_n\}$. Множество также может быть бесконечным и включать все значения в некотором

интервале. Множество, содержащее все значения между a и b включительно, обозначается $[a, b]$. Если же множество не включает граничные значения a и b , то оно обозначается (a, b) . Например, множество $[0, 1]$ включает среди прочего значения 0, 0.0001, 0.25, 0.784, 0.9995 и 1.0. Буквой \mathbb{R} обозначается множество всех чисел от минус бесконечности до плюс бесконечности.

Если элемент x принадлежит множеству \mathcal{S} , то мы пишем $x \in \mathcal{S}$. Новое множество \mathcal{S}_3 можно получить как **пересечение** двух множеств \mathcal{S}_1 и \mathcal{S}_2 . В этом случае мы пишем $\mathcal{S}_3 \leftarrow \mathcal{S}_1 \cap \mathcal{S}_2$. Например, $\{1, 3, 5, 8\} \cap \{1, 8, 4\}$ дает новое множество $\{1, 8\}$.

Новое множество \mathcal{S}_3 можно получить как **объединение** двух множеств \mathcal{S}_1 и \mathcal{S}_2 . В этом случае мы пишем $\mathcal{S}_3 \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2$. Например, $\{1, 3, 5, 8\} \cup \{1, 8, 4\}$ дает новое множество $\{1, 3, 5, 8, 4\}$.

$|\mathcal{S}|$ обозначает размер множества \mathcal{S} , то есть число элементов в нем.

1.1.2. Заглавная сигма

Сумма множества чисел $\mathcal{X} = \{x_1, x_2, \dots, x_{n-1}, x_n\}$ или атрибутов вектора $\mathbf{x} = [x^{(1)}, x^{(2)}, \dots, x^{(m-1)}, x^{(m)}]$ обозначается следующим образом:

$$\sum_{i=1}^n x_i \stackrel{\text{def}}{=} x_1 + x_2 + \dots + x_{n-1} + x_n,$$

или

$$\sum_{i=1}^n x^{(j)} \stackrel{\text{def}}{=} x^{(1)} + x^{(2)} + \dots + x^{(m-1)} + x^{(m)}.$$

Здесь $\stackrel{\text{def}}{=}$ означает «по определению равно».

Евклидова норма вектора \mathbf{x} , обозначаемая $\|\mathbf{x}\|$, характеризует «размер», или «длину», вектора. Она определяется как $\sqrt{\sum_{j=1}^D (x^{(j)})^2}$.

В качестве расстояния между векторами \mathbf{a} и \mathbf{b} берется **евклидово расстояние**:

$$\|\mathbf{a} - \mathbf{b}\| \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^N (a^{(i)} - b^{(i)})^2}.$$

1.2. Что такое машинное обучение

Машинное обучение – это раздел информатики, посвященный построению алгоритмов, которые работают с набором примеров, описывающих какое-то явление. Примеры могут поступать из природы, создаваться людьми или генерироваться другим алгоритмом.

Машинное обучение также можно определить как процесс решения практической задачи путем

- 1) сбора набора данных и
- 2) алгоритмического обучения **статистической модели** на этом наборе.

Предполагается, что эта статистическая модель каким-то образом используется для решения практической задачи. Для краткости я буду использовать термины «обучение» и «машинное обучение» как синонимы. По той же причине я буду часто говорить «модель», имея в виду статистическую модель.

Обучение бывает с учителем, без учителя, с частичным привлечением учителя и с подкреплением.

1.2.1. Обучение с учителем

В случае **обучения с учителем** аналитик работает с набором **помеченных примеров** $\{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$. Каждый элемент x_i называется **вектором признаков**. В информатике вектор – это одномерный массив. Одномерный массив, в свою очередь, является упорядоченной и индексированной последовательностью значений. Длина этой последовательности значений, D , называется **размерностью** вектора.

Вектор признаков – это вектор, в котором каждое измерение $j = 1 \dots D$ содержит значение, описывающее пример. Каждое такое значение называется **признаком** и обозначается $x^{(j)}$. Например, если каждый пример x представляет человека, то первый признак, $x^{(1)}$, может содержать рост в сантиметрах, второй признак, $x^{(2)}$, – вес в килограммах, $x^{(3)}$ – пол и т. д. Для всех примеров в наборе данных значение в позиции j вектора признаков всегда имеет один и тот же тип. Это означает, что если $x_i^{(2)}$ содержит вес в килограммах для некоторого i , то и для любого k от 1 до N $x_k^{(2)}$ будет содержать вес в килограммах. **Метка** y_i может быть либо элементом конечного множества классов $\{1, 2, \dots, C\}$, либо вещественным числом, либо более сложной структурой, такой как вектор, матрица, дерево или граф. Если явно не оговорено противное, то в этой книге y_i является либо элементом конечного множества классов, либо вещественным числом¹. Можно считать, что класс – это категория, к которой относится пример.

Скажем, если примерами являются сообщения электронной почты, а наша задача заключается в обнаружении спама, то есть два класса: спам и не спам. В случае обучения с учителем задача предсказания класса называется **классификацией**, а задача предсказания вещественного числа называется **регрессией**. Значение, которое должно быть предсказано моделью, обученной с учителем, называется **целевым показателем**, или целью. Примером регрессии является задача предсказания заработной платы сотрудника с учетом его опыта работы и знаний. Примером классификации является ситуация, когда врач вводит характеристики пациента в приложение, а то возвращает диагноз.

Различие между классификацией и регрессией показано на рис. 1.2. В случае классификации алгоритм обучения ищет линию (или, в более общем слу-

¹ Вещественное число – это величина, представляющая расстояние вдоль прямой от некоторой начальной точки на ней. Примеры: 0, -256.34, 1000, 1000.2.

чае, гиперповерхность), которая разделяет примеры разных классов. С другой стороны, в случае регрессии алгоритм обучения стремится отыскать линию или гиперповерхность, которая хорошо соответствует обучающим примерам.

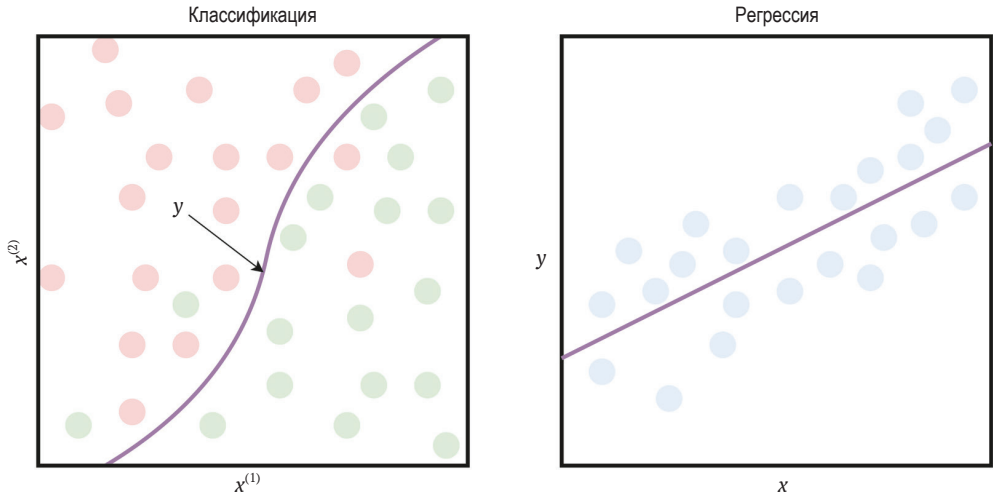


Рис. 1.2 ❖ Разница между классификацией и регрессией

Цель **алгоритма обучения с учителем** – использовать набор данных для порождения модели, которая на входе принимает вектор признаков \mathbf{x} , а на выходе выдает информацию, позволяющую вывести метку для этого вектора признаков. Например, модель, созданная с использованием набора данных о пациентах, может на входе принимать вектор признаков, описывающий пациента, и на выходе выдавать вероятность наличия у пациента рака.

Даже если модель традиционно представляет собой математическую функцию, размышляя о действиях модели с входными данными, удобно думать, что модель «смотрит» на значения некоторых признаков на входе и, основываясь на опыте работы с аналогичными примерами, выводит значение. Это выходное значение представляет собой число или класс, «наиболее похожий» на метки, которые мы видели в прошлом в примерах с похожими значениями признаков. Такое описание выглядит упрощенно, но именно так работает модель решающего дерева и алгоритм k ближайших соседей.

1.2.2. Обучение без учителя

В случае **обучения без учителя** набор данных содержит **непомеченные примеры** $\{x_1, x_2, \dots, x_N\}$. Как и раньше, \mathbf{x} – вектор признаков, а цель **алгоритма обучения без учителя** заключается в порождении модели, которая на входе принимает вектор признаков \mathbf{x} и преобразовывает его либо в другой вектор, либо в значение, используемое для решения практической задачи. Например, в случае **кластеризации** для каждого вектора признаков из на-

бора данных модель возвращает ИД кластера. Кластеризация применяется для отыскания групп похожих объектов в большом наборе объектов, например изображений или текстовых документов. Используя кластеризацию, аналитик может, например, выбрать достаточно репрезентативное, но малое подмножество непомеченных примеров из большого набора, чтобы потом пометить их вручную: из каждого кластера выбирается всего несколько примеров, вместо того чтобы отбирать непосредственно из большого набора с риском выбрать очень похожие примеры.

В задаче **понижения размерности** выходом модели является вектор признаков с меньшим числом измерений, чем на входе. Например, у исследователя имеется вектор признаков, слишком сложный для визуализации (поскольку число измерений больше трех). Модель понижения размерности может преобразовать этот вектор в другой (сохраняя часть информации) – двумерный или трехмерный. Этот новый вектор признаков можно изобразить на графике.

В задаче **обнаружения выбросов** выходом является действительное число, показывающее, насколько входной вектор признаков отличается от «типичного» примера в наборе данных. Обнаружение выбросов применяется для решения задачи о проникновении в сеть (путем обнаружения аномальных сетевых пакетов, которые отличаются от типичного пакета в «нормальном» трафике) или обнаружения новизны (например, документа, отличающегося от других документов в наборе).

1.2.3. Обучение с частичным привлечением учителя

В случае **обучения с частичным привлечением учителя** набор данных содержит как помеченные, так и непомеченные примеры. Обычно число непомеченных примеров намного превышает число помеченных. Цель **алгоритма с частичным привлечением учителя** такая же, как у алгоритма обучения с учителем. Мы надеемся, что, располагая большим числом непомеченных примеров, алгоритм обучения сможет отыскать (можно было бы сказать «породить» или «вычислить») лучшую модель.

1.2.4. Обучение с подкреплением

Обучение с подкреплением – это раздел машинного обучения, в котором рассматривается ситуация, когда машина (именуемая агентом) «обитает» в окружающей среде и способна воспринимать состояние этой среды как вектор признаков. Машина может выполнять действия в незаключительных состояниях. Разные действия приносят разные вознаграждения, а также могут переводить машину в другое состояние окружающей среды. Типичная цель алгоритма обучения с подкреплением – обучиться оптимальной **политике**.

Оптимальная политика – это функция (аналогичная модели в обучении с учителем), которая на входе принимает вектор признаков состояния, а на

выходе выдает оптимальное действие, которое следует выполнить в этом состоянии. Действие является оптимальным, если оно максимизирует ожидаемое среднее долгосрочное вознаграждение.

Обучение с подкреплением решает конкретную задачу, в которой принятие решений является последовательным, а цель – долгосрочной. Такие задачи возникают в играх, робототехнике, управлении ресурсами или логистике.

В этой книге мы для простоты ограничиваемся обучением с учителем (в большинстве случаев). Однако весь представленный в книге материал применим и к другим видам машинного обучения.

1.3. ТЕРМИНОЛОГИЯ МАШИННОГО ОБУЧЕНИЯ

Теперь познакомимся с общепринятой терминологией, относящейся к самим данным (например, данные, используемые прямо и косвенно, первичные и аккуратные данные, обучающие и зарезервированные данные) и к машинному обучению (например, ориентир, гиперпараметр, конвейер и др.).

1.3.1. Данные, используемые прямо и косвенно

Данные в проекте машинного обучения могут использоваться для формирования примеров **x** **прямо** или **косвенно**.

Допустим, что мы строим систему распознавания именованных сущностей. На вход модели поступает последовательность слов, на выходе выдается последовательность меток¹ той же длины, что вход. Для того чтобы алгоритм машинного обучения мог прочитать данные, мы должны преобразовать каждое слово естественного языка в машиночитаемый массив атрибутов, называемый вектором признаков². Некоторые признаки могут содержать информацию, которая отличает это конкретное слово от других слов в словаре. Другие признаки могут содержать дополнительные атрибуты слова в этой конкретной последовательности, такие как его форма (в нижнем регистре, в верхнем регистре, с заглавной буквы и т. д.). Либо это могут быть бинарные атрибуты, указывающие на то, является ли слово первым словом имени человека или последним словом названия какого-либо места или организации. Для создания таких бинарных признаков мы можем прибегнуть к словарям, справочным таблицам, географическим справочникам или другим моделям машинного обучения, делающим предсказания относительно слов.

Вы, вероятно, уже поняли, что набор последовательностей слов – это данные, используемые для формирования обучающих примеров непосредствен-

¹ Метки могут быть, например, значениями из множества {«Местоположение», «Организация», «Лицо», «Прочее»}.

² Термины «атрибут» и «признак» часто используются как синонимы. В этой книге я применяю термин «атрибут» для описания конкретного свойства примера, тогда как термин «признак» относится к значению $x^{(j)}$ в позиции j вектора признаков \mathbf{x} , используемого алгоритмом машинного обучения.

но, тогда как данные, содержащиеся в словарях, справочных таблицах и географических справочниках, используются косвенно: они могут служить для пополнения векторов признаков дополнительными признаками, но не для создания новых векторов.

1.3.2. Первичные и аккуратные данные

Как мы только что обсудили, непосредственно используемые данные – это набор сущностей, составляющих основу набора данных. Каждая сущность в этом наборе может быть преобразована в обучающий пример. **Первичные данные** – это набор сущностей в их естественной форме; их не всегда можно задействовать для машинного обучения непосредственно. Например, документ Word или JPEG-файл являются первичными данными; алгоритм машинного обучения не может использовать их напрямую¹.

Необходимым (но не достаточным) условием использования данных в машинном обучении является их аккуратность. **Аккуратные данные** можно рассматривать как электронную таблицу, в которой строки представляют примеры, а столбцы – **атрибуты** примеров, как показано на рис. 1.3. Иногда сами первичные данные могут быть аккуратными, например если они уже представлены в виде электронной таблицы. Однако на практике, чтобы получить аккуратные данные из первичных, аналитики нередко прибегают к процедуре **конструирования признаков**, которая применяется к прямым и, факультативно, к косвенным данным с целью преобразования каждого первичного примера в вектор признаков x . Глава 4 полностью посвящена конструированию признаков.

Атрибуты			
Country	Population	Region	GDP
France	67M	Europe	2.8T
Germany	83M	Europe	3.7T
...
China	1396M	Asia	12.2T

Примеры			
Country	Population	Region	GDP
France	67M	Europe	2.8T
Germany	83M	Europe	3.7T
...
China	1396M	Asia	12.2T

Рис. 1.3 ❖ Аккуратные данные:
строки содержат примеры, а столбцы – атрибуты

¹ Термин «неструктурированные данные» часто используется для обозначения элемента данных, содержащего информацию, тип которой формально не определен. Примерами неструктурированных данных являются фотографии, изображения, видео, текстовые сообщения, сообщения в социальных сетях, PDF-файлы, текстовые документы и электронные письма. Термин «полуструктурированные данные» относится к элементам данных, структура которых помогает выводить типы некоторой закодированной в них информации. К полуструктурированным данным можно отнести файлы журналов, текстовые файлы с запятыми или табуляторами в качестве разделителей, а также документы в форматах JSON и XML.

Здесь важно отметить, что в некоторых задачах пример, используемый алгоритмом обучения, может быть последовательностью векторов, матрицей или последовательностью матриц. Понятие аккуратности данных для таких алгоритмов определяется в схожем ключе: нужно лишь заменить «строку фиксированной ширины в электронной таблице» матрицей фиксированной ширины и высоты либо обобщением матрицы на большее число измерений – **тензором**.

Термин «аккуратные данные» был введен Хэдли Уикхемом в одноименной статье¹.

Как было сказано в начале этого раздела, данные могут быть аккуратными, но все равно непригодными для использования конкретным алгоритмом машинного обучения. На самом деле большинство алгоритмов машинного обучения принимают обучающие данные только в виде набора векторов числовых признаков. Возьмем данные, показанные на рис. 1.3. Атрибут «Регион» является категориальным, а не числовым. Алгоритм обучения на основе решающего дерева может работать со значениями категориальных атрибутов, но большинство алгоритмов обучения на это не способны. В разделе 4.2 главы 4 мы увидим, как преобразовать категориальный атрибут в числовой.

Обратите внимание, что в академической литературе по машинному обучению слово «пример» обычно относится к примеру аккуратных данных, возможно, с соответственной меткой. Однако на этапе сбора данных и их разметки, который мы рассмотрим в следующей главе, примеры могут быть еще представлены в первичной форме: изображения, тексты или строки с категориальными атрибутами в электронной таблице. В этой книге, когда важно подчеркнуть разницу, я буду говорить «**первичный пример**», чтобы указать, что часть данных еще не была преобразована в вектор признаков. В противном случае мы будем предполагать, что примеры имеют форму векторов признаков.

1.3.3. Обучающие и зарезервированные наборы

На практике аналитики работают с тремя разными наборами примеров:

- 1) обучающий набор;
- 2) контрольный набор²;
- 3) тестовый набор.

Получив данные в виде набора примеров, первое, что нужно сделать в проекте, – перемешать примеры и разделить данные на три отдельных набора: **обучающий, контрольный и тестовый**. Обучающий набор обычно является самым крупным; он используется в алгоритме обучения для порождения модели. Контрольный и тестовый наборы имеют примерно одинаковый, гораздо меньший, размер. Алгоритму обучения не разрешается использовать

¹ Wickham Hadley. Tidy data // Journal of Statistical Software 59.10 (2014): 1–23.

² Иногда, если помеченных примеров недостаточно, аналитик может обойтись без контрольного набора, как мы увидим в разделе главы 5, посвященном **перекрестной проверке**.

примеры из контрольного или тестового набора для обучения модели. Поэтому оба этих набора иногда называют **зарезервированными**.

Причина наличия трех наборов, а не одного, проста: в процессе обучения мы не хотим, чтобы модель хорошо предсказывала метки только тех примеров, которые алгоритм уже встречал. Тривиальный алгоритм, который просто запоминает все обучающие примеры, а затем использует их для «предсказания» меток, не будет делать никаких ошибок, когда его попросят предсказать метки обучающих примеров. Однако на практике такой алгоритм будет бесполезен. Действительно, нам нужна модель, хорошо предсказывающая примеры, которые не предъявлялись алгоритму во время обучения. Иными словами, мы хотим иметь хорошее качество на зарезервированном наборе¹.

Два зарезервированных набора, а не один, нужно, потому что контрольный набор используется, чтобы 1) выбирать алгоритм обучения и 2) отыскивать оптимальные конфигурационные параметры для этого алгоритма (именуемые **гиперпараметрами**). Тестовый набор применяется для оценивания модели перед ее передачей заказчику или развертыванием в производственной среде. Вот почему так важно, чтобы никакая информация из контрольного или тестового набора не была предъявлена алгоритму обучения. В противном случае результаты контроля и тестирования, скорее всего, будут слишком оптимистичными. Такая неприятность иногда случается из-за **просачивания данных**, явления, которое мы рассмотрим в разделе 3.2.8 главы 3 и последующих главах.

1.3.4. Ориентир

В машинном обучении **ориентиром** называется простой алгоритм решения задачи, обычно основанный на эвристике, простой сводной статистике, рандомизации или очень простом алгоритме машинного обучения. Например, если задача связана с классификацией, то можно выбрать классификатор, служащий ориентиром, и измерить его качество. С этим ориентиром затем будет сравниваться любая будущая модель (обычно построенная с применением более изощренного подхода).

1.3.5. Конвейер машинного обучения

Конвейер машинного обучения – это последовательность операций с набором данных, ведущая из начального состояния к модели.

Конвейер может включать, среди прочего, такие этапы, как разбиение данных, подстановка пропущенных данных, выделение признаков, приращение

¹ Если быть точным, мы хотим, чтобы модель хорошо работала на большинстве случайных выборок из статистического распределения, которому принадлежат наши данные. Мы исходим из того, что если модель хорошо работает на зарезервированном наборе, случайно выбранном из неизвестного распределения данных, то высоки шансы, что она будет хорошо работать и на других случайно выбранных примерах.

данных, уменьшение несбалансированности классов, понижение размерности и обучение модели.

На практике, развертывая модель в производственной среде, мы обычно развертываем весь конвейер. Кроме того, при настройке гиперпараметров обычно оптимизируется весь конвейер целиком.

1.3.6. Параметры и гиперпараметры

Гиперпараметры – это входные данные алгоритма машинного обучения или конвейера, которые влияют на качество модели. Они не относятся к обучающим данным и не могут быть обучены на их основе. Например, максимальная глубина дерева в алгоритме решающего дерева, штраф за неправильную классификацию в методе опорных векторов, величина k в алгоритме k ближайших соседей, целевая размерность в алгоритме понижения размерности и выбор способа подстановки пропущенных данных – все это примеры гиперпараметров.

Параметры, с другой стороны, являются переменными, которые определяют обучаемую модель. Параметры напрямую модифицируются алгоритмом обучения на основе обучающих данных. Цель обучения состоит в том, чтобы отыскать такие значения параметров, которые делают модель в определенном смысле оптимальной. Примерами параметров являются w и b в уравнении линейной регрессии $y = wx + b$, где x является входом модели, а y – ее выходом (предсказанием).

1.3.7. Классификация и регрессия

Классификация – это задача автоматического назначения метки **непомеченному примеру**. Распознавание спама – широко известный пример классификации.

В машинном обучении задача классификации решается **алгоритмом обучения классификатора**, который на входе принимает набор **помеченных примеров**, а на выходе порождает **модель**, которая может получать непомеченный пример и возвращать либо саму метку, либо число, которое аналитик может использовать для нахождения метки. Примером такого числа является вероятность того, что элемент входных данных имеет определенную метку.

В задаче классификации метка является элементом конечного множества **классов**. Если размер этого множества равен двум (больной/здоровый, спам/не спам), то мы говорим о **бинарной классификации** (в некоторых источниках она называется **биномиальной**). **Многоклассовая** (или **мультиномиальная**) **классификация** – это задача классификации с тремя или более классами¹.

В то время как некоторые алгоритмы обучения естественным образом допускают наличие более двух классов, другие по природе своей являются

¹ Тем не менее каждому примеру по-прежнему соответствует одна метка.

алгоритмами бинарной классификации. Существуют стратегии преобразования алгоритма бинарной классификации в многоклассовый. Об одном из них, «один против остальных», я расскажу в разделе 6.5 главы 6.

Регрессия – это задача предсказания вещественной величины по непомеченному примеру. Широко известный пример регрессии – оценка стоимости дома на основе таких его характеристик, как площадь, число спален, местоположение и т. д.

Задача регрессии решается с помощью **алгоритма обучения регрессии**; на входе он принимает набор помеченных примеров и порождает модель, которая принимает непомеченный пример и выводит целевой показатель.

1.3.8. Обучение на основе модели и обучение на основе экземпляров

Большинство алгоритмов обучения с учителем **основаны на моделях**. Типичной **моделью** является **метод опорных векторов (Support Vector Machine – SVM)**. В алгоритмах обучения на основе моделей обучающие данные используются для создания модели с обученными **параметрами**. В SVM таких параметров два: w (вектор) и b (вещественное число). Обученную модель можно сохранить на диске, а обучающие данные удалить.

В **алгоритмах обучения на основе экземпляров** весь набор данных целиком используется в качестве модели. Одним из часто используемых на практике алгоритмов такого рода является метод **k ближайших соседей (kNN)**. Для предсказания метки входного примера алгоритм kNN рассматривает его окрестность в пространстве векторов признаков и выводит метку, которая встречается в этой окрестности чаще других.

1.3.9. Поверхностное и глубокое обучение

Алгоритм **поверхностного обучения** обучает параметры непосредственно на признаках обучающих примеров. Большинство алгоритмов машинного обучения поверхностные. Широко известными исключениями являются **нейросетевые** алгоритмы обучения, а именно такие, которые строят нейронные сети с несколькими **слоями** между входом и выходом. Такие сети называются **глубокими нейронными сетями**. В глубоком нейросетевом обучении (или просто **глубоком обучении**), в отличие от поверхностного, большинство параметров модели обучаются не на самих признаках обучающих примеров, а на выходах предыдущих слоев.

1.3.10. Обучение и оценивание

Когда алгоритм машинного обучения применяется к набору данных с целью получения модели, говорят об **обучении модели** или просто обучении.

Когда обученная модель применяется к входному примеру (иногда к последовательности примеров) с целью получить предсказание (или несколько

предсказаний) либо каким-то образом преобразовать входные данные, говорят об **оценивании**.

1.4. Когда следует использовать МАШИННОЕ ОБУЧЕНИЕ

Машинное обучение – мощный инструмент для решения практических задач. Однако, как и любой инструмент, его следует использовать в правильном контексте. Попробовать решать все задачи с помощью машинного обучения было бы ошибкой.

Возможность использования машинного обучения следует рассматривать в одной из следующих ниже ситуаций.

1.4.1. Когда задача слишком сложна для кодирования

В ситуации, когда задача настолько сложна или велика, что попытка написать все правила для ее решения попросту безнадежна, а частичное решение допустимо и представляет интерес, можно попытаться решить задачу с помощью машинного обучения.

Один из примеров – обнаружение спама: невозможно написать исходный код, реализующий алгоритм, который будет эффективно обнаруживать спамные сообщения и помещать в папку входящих только хорошие сообщения. Придется учитывать слишком много факторов. Например, если запрограммировать спам-фильтр на отклонение всех сообщений от лиц, отсутствующих в контактах, то имеется риск потерять сообщения от человека, получившего вашу визитную карточку на конференции. Сделав исключение для сообщений, содержащих определенные ключевые слова, связанные с работой, вы, скорее всего, пропустите сообщение от учителя своего ребенка и т. д.

Если вы все же решите запрограммировать решение этой сложной задачи в лоб, то со временем в исходном коде окажется так много правил и исключений из них, что сопровождать этот код станет невозможно. В этой ситуации обучение классификатора на примерах спама и не спама кажется логичным и единственно жизнеспособным вариантом.

Еще одна трудность при написании исходного кода для решения задачи – тот факт, что людям трудно решать задачи прогнозирования, основываясь на входных данных, содержащих слишком много параметров, особенно если параметры **коррелируют** неизвестным образом. Возьмем, к примеру, предсказание возможности погашения кредита заемщиком. Каждый заемщик представлен сотнями показателей: возраст, зарплата, остаток на счете, частота прошлых платежей, семейное положение, число детей, марка и год выпуска автомобиля, остаток по ипотеке и т. д. Одни из них могут быть важны для принятия решения, другие сами по себе менее важны, но становятся более важными, если рассматривать их в сочетании.

Писать исходный код, который будет принимать такие решения, трудно, потому что даже эксперту не ясно, как оптимально объединять в предсказание все описывающие человека атрибуты.

1.4.2. Когда задача постоянно меняется

Некоторые задачи со временем постоянно изменяются, поэтому исходный код необходимо регулярно обновлять. Это «бесит» программистов, работающих над задачей, растет вероятность ошибок, становится трудно совмещать «старую» логику с «новой», и значительно возрастают накладные расходы на тестирование и развертывание обновленных решений.

Например, рассмотрим задачу выделения представляющих интерес элементов данных из коллекции веб-страниц. Предположим, что для каждой веб-страницы создается фиксированный набор правил извлечения данных в следующей форме: «выбрать третий элемент `<p>` из `<body>`, а затем выбрать данные из второго `<div>` внутри этого `<p>`». Если владелец сайта изменит дизайн страницы, то интересующие вас данные могут оказаться во втором или четвертом элементе `<p>`, и правило извлечения станет недействительным. Если коллекция обрабатываемых страниц велика (тысячи URL-адресов), то недействительные правила будут возникать каждый день, так что процесс их настройки никогда не закончится. Излишне говорить, что очень немногие программисты хотели бы выполнять такую работу ежедневно.

1.4.3. Когда речь идет о задаче восприятия

Сегодня трудно представить себе, что кто-то попытается решать **задачи восприятия**, такие как распознавание речи, изображений и видео, без использования машинного обучения. Возьмем, к примеру, изображение. Оно представлено миллионами пикселей. Каждый пиксель обозначается тремя числами: интенсивностью красного, зеленого и синего каналов. В прошлом инженеры пытались решать задачу распознавания изображений (определения того, что изображено на картинке) путем применения самодельных «фильтров» к квадратным участкам пикселей – патчам. Если, например, фильтр, предназначенный для «обнаружения» травы, порождает большое значение при применении ко многим патчам, а другой фильтр, предназначенный для обнаружения коричневого подшерстка, тоже возвращает большие значения для многих патчей, то высоки шансы, что изображена корова на лугу (я немного упрощаю).

Сегодня задачи восприятия эффективно решаются с помощью моделей машинного обучения, таких как нейронные сети. Мы рассмотрим задачу обучения нейронных сетей в главе 6.

1.4.4. Когда это неизученное явление

Если требуется предсказывать явление, которое недостаточно изучено с научной точки зрения, но наблюдаемо, то машинное обучение, возможно, будет

подходящим (а в некоторых случаях единственно доступным) вариантом. Например, машинное обучение можно использовать для разработки персонализированных способов лечения психических расстройств на основе генетических и сенсорных данных. Врачи не всегда могут интерпретировать такие данные и дать оптимальную рекомендацию, тогда как машина способна обнаружить в данных закономерности благодаря анализу тысяч пациентов и предсказать молекулу, которая имеет наибольшие шансы помочь данному пациенту.

Еще один пример наблюдаемых, но неизученных явлений – журналы сложной вычислительной системы или сети. Такие журналы создаются несколькими независимыми или взаимозависимыми процессами. Человеку трудно делать предсказания о будущем состоянии системы, располагая только журналами, но не имея модели каждого процесса и их взаимозависимостей. Если количество исторических журнальных записей достаточно велико (как часто и бывает), то машина сможет выявлять скрытые закономерности и делать предсказания, ничего не зная о каждом процессе.

Наконец, трудно делать прогнозы о людях, основываясь на их наблюдаемом поведении. В этой задаче у нас, очевидно, отсутствует модель мозга человека, но имеются легкодоступные примеры выражения его мыслей (в виде онлайн-сообщений, комментариев и других действий). Основываясь только на этих выражениях, развернутая в социальной сети модель машинного обучения сможет рекомендовать контент или других людей для общения.

1.4.5. Когда задача имеет простую целевую функцию

Машинное обучение особенно хорошо подходит в случаях, когда задачу можно сформулировать как оптимизацию простой целевой функции: например, когда требуется найти бинарное решение типа да/нет или одно-единственное число. С другой стороны, машинное обучение не подойдет для построения модели, работающей как типичная видеоигра, например Mario, или как текстовый процессор типа Word. Дело в том, что число принимаемых решений слишком велико: что, где и когда выводить на экран, как система должна реагировать на введенные пользователем данные, что записывать или читать с жесткого диска и т. д.; получить примеры, иллюстрирующие все (или даже большинство) из этих решений, практически невозможно.

1.4.6. Когда это экономически выгодно

Три основных источника затрат в машинном обучении таковы:

- сбор, подготовка и очистка данных;
- обучение модели;
- создание и эксплуатация инфраструктуры для выполнения и мониторинга модели, а также трудовые ресурсы для ее сопровождения.

Стоимость обучения модели включает в себя человеческий труд и в некоторых случаях дорогостоящее оборудование, необходимое для обучения глубоких моделей. Сопровождение модели включает постоянный мониторинг ее работы и сбор дополнительных данных для поддержания модели в актуальном состоянии.

1.5. Когда не следует использовать МАШИННОЕ ОБУЧЕНИЕ

Существует много задач, которые невозможно решить с помощью машинного обучения; трудно охарактеризовать их все. Здесь мы дадим лишь несколько рекомендаций.

Пожалуй, не стоит использовать машинное обучение, когда:

- любое действие системы или принятое ею решение должно быть объяснимым;
- любое изменение в поведении системы по сравнению с ее прошлым поведением в аналогичной ситуации должно быть объяснимым;
- стоимость допущенной системой ошибки слишком высока;
- требуется вывести продукт на рынок как можно быстрее;
- получить правильные данные слишком трудно или невозможно;
- задачу можно решить с помощью традиционных методов разработки программ с меньшими затратами;
- простая эвристика работает достаточно хорошо;
- у явления слишком много исходов, а получить примеры в количестве, достаточном для их представления, невозможно (например, в видеоиграх или текстовых процессорах);
- построенную систему со временем не придется часто улучшать;
- можно вручную подготовить исчерпывающую справочную таблицу, указав ожидаемый результат для любого входного значения (то есть число возможных входных значений не слишком велико или получать выходные данные можно быстро и дешево).

1.6. Что такое ИНЖЕНЕРИЯ МАШИННОГО ОБУЧЕНИЯ

Инженерия машинного обучения (machine learning engineering – MLE) – это использование научных принципов, инструментов и методов машинного обучения и традиционной программной инженерии для проектирования и создания сложных компьютерных систем. MLE охватывает все этапы от сбора данных до обучения модели и использования ее продуктом или клиентами.

Традиционно аналитик данных¹ занимается осмыслением деловой задачи, построением модели для ее решения и оцениванием модели в ограниченной среде разработки. В обязанности инженера по машинному обучению входят получение данных из различных систем и мест, их предобработка, конструирование признаков, обучение эффективной модели, которая будет работать в производственной среде, сосуществовать с другими производственными процессами, будет стабильной, удобной для сопровождения и легкодоступной для разных типов пользователей, по-разному применяющих ее.

Иными словами, MLE включает в себя любую деятельность, которая позволяет реализовать алгоритмы машинного обучения как часть эффективной производственной системы.

На практике инженеры по машинному обучению могут быть задействованы в таких видах деятельности, как переписывание созданного аналитиками данных кода с таких относительно медленных языков, как R и Python², на более эффективных языках типа Java или C++, масштабирование этого исходного кода и повышение его надежности, его упаковка в легко развертываемый пакет, хранящийся в системе управления версиями, оптимизация алгоритма машинного обучения, так чтобы порождаемая им модель была совместима с производственной средой организации и работала в ней правильно.

Во многих организациях аналитики данных выполняют такие задачи MLE, как сбор данных, их преобразование и конструирование признаков. С другой стороны, инженеры по машинному обучению нередко выполняют некоторые задачи анализа данных, включая выбор алгоритма обучения, настройку гиперпараметров и оценивание модели.

Работа над проектом машинного обучения отличается от работы над типичным проектом в области программной инженерии. В отличие от традиционной программной инженерии, где поведение программы обычно детерминировано, приложения машинного обучения включают модели, поведение которых со временем может по естественным причинам ухудшаться или становиться аномальным. Такое аномальное поведение модели может иметь различные корни, в т. ч. фундаментальное изменение входных данных или внедрение нового экстрактора признаков, возвращающего другое распределение значений или значения иного типа. Нередко можно услышать, что системы машинного обучения «отказывают молча». Инженер по машинному обучению должен быть способен предотвращать такие отказы либо, если полное предотвращение невозможно, знать, как их обнаруживать и устранять.

¹ Должность исследователя данных (data scientist) приобрела популярность примерно с 2013 года. К сожалению, компании и эксперты не пришли к единому мнению относительно определения этого термина. Вместо этого я использую термин «аналитик данных», имея в виду человека, способного применять численный или статистический анализ к имеющимся данным.

² Многие научные модули в Python на самом деле написаны на быстром C/C++; однако собственный код аналитика данных на Python все равно может быть медленным.

1.7. ЖИЗНЕННЫЙ ЦИКЛ ПРОЕКТА МАШИННОГО ОБУЧЕНИЯ

Проект машинного обучения начинается с осмысления целевого критерия с точки зрения бизнеса. Обычно бизнес-аналитик работает с заказчиком¹ и аналитиком данных, чтобы трансформировать деловую задачу в технический проект. Технический проект может содержать или не содержать часть, связанную с машинным обучением. В этой книге мы, разумеется, будем рассматривать проекты, в которых машинное обучение используется.

После определения технического проекта как раз и начинается область инженерии машинного обучения. Машинное обучение в рамках более широкого технического проекта прежде всего должно иметь четко определенную **цель**. Целью машинного обучения является описание того, что статистическая модель получает на входе, что она генерирует на выходе, и критериев приемлемого (или неприемлемого) поведения модели.

Цель машинного обучения не обязательно совпадает с бизнес-целью. Бизнес-цель – это то, чего хочет достичь организация. Например, бизнес-цель компании Google в случае Gmail может заключаться в том, чтобы сделать Gmail наиболее популярной почтовой службой в мире. Для достижения этой цели Google может создавать многочисленные технические проекты, включающие машинное обучение. Цель одного из таких проектов может состоять в том, чтобы отличать содержательные электронные письма от рекламных акций с точностью выше 90 %.

В целом жизненный цикл проекта машинного обучения, показанный на рис. 1.4, состоит из следующих этапов: 1) определение цели, 2) сбор и подготовка данных, 3) конструирование признаков, 4) обучение модели, 5) оценивание модели, 6) развертывание модели, 7) выполнение модели, 8) мониторинг модели и 9) сопровождение модели.

На рис. 1.4 область применения машинного обучения (и этой книги) ограничена синей зоной. Сплошные стрелки показывают типичный технологический поток. Пунктирные стрелки означают, что на некоторых этапах может быть принято решение вернуться назад и либо собрать больше данных, либо собрать другие данные и пересмотреть признаки (исключив одни и сконструировав другие).

Каждый из упомянутых этапов будет рассмотрен в одной из глав книги. Но сначала давайте обсудим вопрос о том, как расставить приоритеты в проектах машинного обучения, определить цель проекта и организовать коллектив разработчиков. Этим трем вопросам посвящена следующая глава.

¹ Если проект машинного обучения используется для поддержки продукта, разработанного и продаваемого организацией, то бизнес-аналитик работает с владельцем продукта.

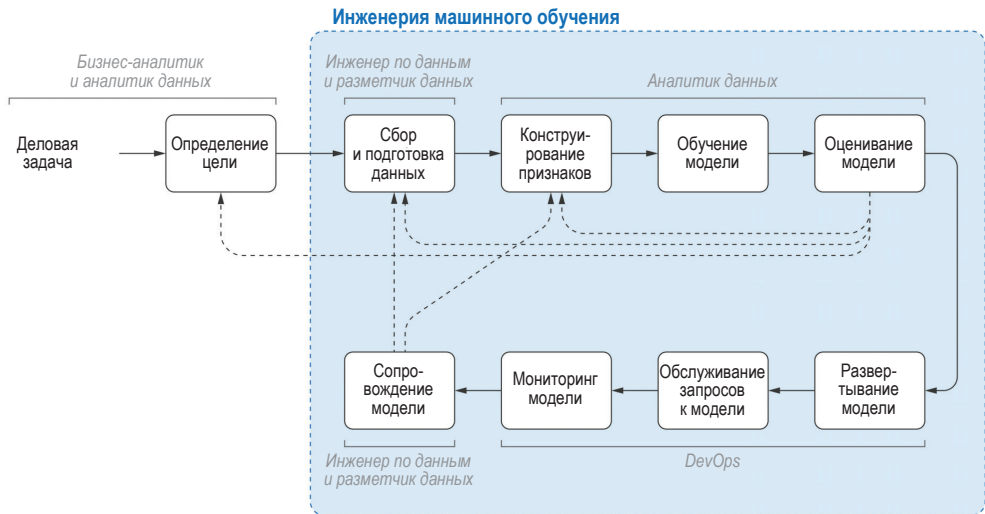


Рис. 1.4 ❖ Жизненный цикл проекта машинного обучения

1.8. РЕЗЮМЕ

Алгоритм машинного обучения на основе модели принимает на входе набор обучающих примеров и выдает на выходе модель. Алгоритм машинного обучения на основе экземпляров использует весь обучающий набор данных в качестве модели. Обучающие данные предъявляются алгоритму машинного обучения, а зарезервированные данные – нет.

Алгоритм обучения с учителем строит модель, которая принимает вектор признаков и выводит для него предсказание. Алгоритм обучения без учителя строит модель, которая принимает вектор признаков и преобразовывает его во что-то полезное.

Классификация – это задача предсказания для входного примера одного из конечного множества классов.

В задачу регрессии входит предсказание числового целевого показателя.

Данные могут использоваться прямо либо косвенно. Прямо используемые данные являются основой для формирования набора примеров. Косвенно используемые данные используются для обогащения этих примеров.

Данные для машинного обучения должны быть аккуратными. Аккуратный набор данных можно трактовать как электронную таблицу, в которой строки представляют примеры, а столбцы – свойства примеров. Помимо аккуратности, в большинстве алгоритмов машинного обучения данные должны быть числовыми, а не категориальными. Конструирование признаков – это процесс преобразования данных в форму, которая может использоваться алгоритмами машинного обучения.

Ориентир необходим для того, чтобы модель могла работать лучше, чем простая эвристика.

На практике машинное обучение реализуется в виде конвейера, содержащего последовательные этапы преобразования данных, от разбиения и подстановки пропущенных данных до устранения несбалансированности классов, понижения размерности и обучения модели. Гиперпараметры всего конвейера обычно оптимизируются; конвейер можно развернуть и использовать для предсказаний.

Параметры модели оптимизируются алгоритмом обучения на основе тренировочных данных. Значения гиперпараметров не обучаются алгоритмически, а настраиваются с помощью контрольного набора данных. Тестовый набор используется только для оценивания качества модели и предоставления отчета о ней клиенту или владельцу продукта.

Алгоритм поверхностного обучения обучает модель, которая делает предсказания непосредственно по входным данным. Алгоритм глубокого обучения обучает многослойную модель, в которой каждый слой генерирует выходные данные, принимая на входе результаты предыдущего слоя.

Рассматривать возможность использования машинного обучения для решения деловой задачи следует, когда задачу слишком сложно закодировать, задача постоянно меняется, задача подразумевает восприятие, представляет неизученное явление, имеет простую целевую функцию и является экономически эффективной.

Во многих ситуациях машинное обучение использовать не стоит: когда требуется объяснимость, когда ошибки недопустимы, когда традиционная программная инженерия менее затратна, когда все входные и выходные данные могут быть перечислены и сохранены в базе данных, а также когда данные трудно или слишком дорого собрать.

Инженерия машинного обучения (MLE) – это использование научных принципов, инструментов и методов машинного обучения и традиционной программной инженерии для проектирования и построения сложных компьютерных систем. MLE охватывает все этапы от сбора данных до обучения модели и использования ее продуктом или потребителями.

Жизненный цикл проекта машинного обучения состоит из следующих этапов: 1) определение цели, 2) сбор и подготовка данных, 3) конструирование признаков, 4) обучение модели, 5) оценивание модели, 6) развертывание модели, 7) выполнение модели, 8) мониторинг модели и 9) сопровождение модели.

Каждый этап будет рассмотрен в одной из глав книги.

Глава 2

Прежде чем приступить к проекту

Прежде чем приступить к проекту машинного обучения, необходимо расставить приоритеты. Расстановка приоритетов неизбежна: возможности коллектива разработчиков и оборудования ограничены, а количество проектов в работе бывает очень большим.

При определении приоритетов оценивается сложность проекта. В машинном обучении редко можно получить точную оценку сложности из-за наличия таких важных неизвестных, как принципиальная достижимость требуемого качества модели, объем необходимых данных, виды и число необходимых признаков.

Кроме того, проект машинного обучения должен иметь четко определенную цель. Исходя из цели проекта, подбирается коллектив разработчиков и решается вопрос об обеспечении его ресурсами.

В этой главе мы рассмотрим эти и прочие действия, которые необходимо выполнить до начала проекта машинного обучения.

2.1. ОПРЕДЕЛЕНИЕ ПРИОРИТЕТОВ ПРОЕКТА МАШИННОГО ОБУЧЕНИЯ

Ключевыми соображениями при определении приоритетов проекта машинного обучения являются его предполагаемые последствия и стоимость.

2.1.1. Последствия машинного обучения

Последствия применения машинного обучения в более широком техническом проекте велики, когда 1) машинное обучение может заменить сложную часть инженерного проекта либо 2) получение недорогих (но, вероятно, несовершенных) предсказаний дает значительные преимущества.

Например, сложная часть существующей системы может быть основана на правилах со множеством вложенных правил и исключений. Процесс по-

строения и сопровождения такой системы бывает чрезвычайно сложным, трудоемким и подвержен ошибкам. К тому же программисты не испытывают никакой радости от сопровождения этой части системы. Можно ли обучить, а не программировать правила? Можно ли использовать существующую систему для генерирования помеченных данных? Если да, то такой проект машинного обучения будет иметь высокую отдачу и низкую стоимость.

Недорогие и несовершенные предсказания бывают полезны, например, в системе, которая обрабатывает большое число запросов. Допустим, многие из таких запросов «легкие», так что на них можно быстро ответить с помощью существующей автоматической подсистемы. Остальные запросы считаются «трудными», так что решать их приходится вручную.

Основанная на машинном обучении система, которая распознает «легкие» задачи и отправляет их автоматической подсистеме, экономит много времени людям, которые будут тратить усилия и время на решение трудных вопросов. Даже если диспетчер допустит ошибку в предсказании, сложный запрос попадет автоматической подсистеме, она с ним не справится, и в конечном итоге запрос будет передан человеку. Если человек получает легкий запрос по ошибке, тоже не страшно: он может быть отправлен автоматической подсистеме или обработан человеком.

2.1.2. Стоимость машинного обучения

На стоимость проекта машинного обучения влияют три фактора:

- сложность задачи,
- стоимость данных и
- необходимая верность.

Получить нужные данные в нужном объеме бывает очень дорого, в особенности если речь идет о ручной разметке. Потребность в высокой точности может означать, что необходимо собрать больше данных или обучить более сложную модель, например глубокую нейронную сеть с уникальной **архитектурой** или модель с нетривиальной **ансамблевой архитектурой**.

При обдумывании сложности задачи на первый план выходят следующие соображения:

- существует ли способный решить задачу готовый алгоритм или программная библиотека (если да, то задача значительно упрощается);
- требуется ли для построения модели или ее выполнения в производственной среде значительная вычислительная мощность.

Вторым определяющим фактором затрат являются данные. Необходимо принимать во внимание следующие аспекты:

- могут ли данные генерироваться автоматически (если да, то задача значительно упрощается);
- какова стоимость ручного **аннотирования** данных (т. е. сопоставления меток непомеченным примерам);
- сколько примеров необходимо (обычно это неизвестно заранее, но может быть оценено на основе известных опубликованных результатов или собственного опыта предприятия).

Наконец, еще одним из наиболее важных факторов стоимости является желаемая верность модели. Стоимость проекта машинного обучения растет сверхлинейно при увеличении требований к верности, как показано на рис. 2.1. Кроме того, низкая верность бывает источником значительных потерь при развертывании модели в производственной среде. При этом необходимо учесть следующие моменты:

- насколько дорого обходится каждое неверное предсказание;
- ниже какого уровня верности модель становится практически бесполезной.

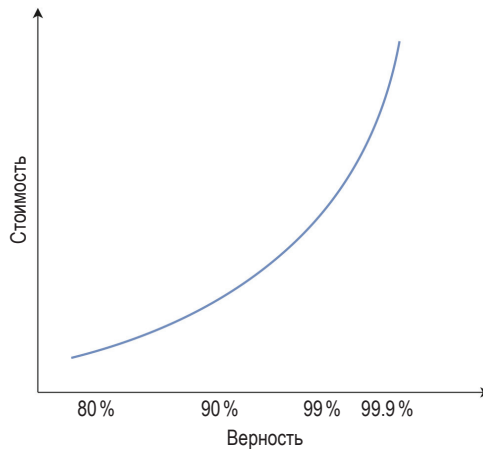


Рис. 2.1 ❖ Сверхлинейный рост стоимости в зависимости от требований к верности

2.2. ОЦЕНИВАНИЕ СЛОЖНОСТИ ПРОЕКТА МАШИННОГО ОБУЧЕНИЯ

Для проекта МО не существует стандартного метода оценивания сложности, кроме сравнения с другими проектами, выполненными организацией или описанными в литературе.

2.2.1. Неизвестные

Существует несколько важных неизвестных, которые почти невозможно угадать с уверенностью, если вы не работали над аналогичным проектом в прошлом или не читали об аналогичных проектах. Вот они:

- достижимо ли требуемое качество на практике;
- сколько данных потребуется для достижения требуемого качества;
- какие и сколько признаков необходимо для того, чтобы обучить модель, которая обобщалась бы в достаточной мере;

- насколько большой должна быть модель (это особенно актуально для нейросетевых и ансамблевых архитектур);
- сколько времени потребуется для обучения одной модели (иными словами, сколько нужно времени для проведения одного **эксперимента**) и сколько экспериментов понадобится для достижения желаемого качества.

Можно быть почти уверенным только в одном: если требуемый уровень **верности** модели (одна из популярных метрик качества модели, которую мы рассматриваем в разделе 5.5 главы 5) превышает 99 %, то можно ожидать осложнений, связанных с недостаточным объемом помеченных данных. В некоторых задачах даже 95%-ная верность считается труднодостижимой. (Здесь мы, конечно же, исходим из того, что данные сбалансированы, то есть отсутствует **несбалансированность классов**, которая будет темой раздела 3.9.)

Еще один полезный ориентир – качество, достигаемое при решении задачи человеком. Обычно очень трудно добиться, чтобы модель работала не хуже человека.

2.2.2. Упрощение задачи

Высказать более обоснованную гипотезу можно, если упростить задачу и сначала решить более простой вариант. Например, предположим, что задачей является классификация множества документов по 1000 тем. Выполните пилотный проект: сосредоточьтесь на 10 темах, а документы, относящиеся к остальным 990 темам, считайте «Прочими»¹. Пометьте данные для этих 11 классов вручную (10 реальных категорий плюс «Прочие»). Логика здесь проста – человеку гораздо проще держать в памяти определения только 10 категорий, а не запоминать различия между 1000 категорий².

Упростив задачу до 11 классов, решите ее и измерьте время на каждом этапе. Убедившись, что задача для 11 классов разрешима, можно обоснованно надеяться, что она будет разрешима и для 1000 классов. Сохраненные измерения затем можно использовать для оценивания времени, необходимого для полного решения задачи, хотя для получения точной оценки нельзя просто умножить время на 100. Объем данных, необходимых для того, чтобы научиться различать много классов, с увеличением числа классов обычно растет сверхлинейно.

Другой способ получения более простой задачи из потенциально сложной – разбить задачу на несколько простых, используя естественные срезы в имеющихся данных. Например, пусть у организации есть клиенты в не-

¹ Помещение примеров, относящихся к 990 классам, в один класс, скорее всего, приведет к созданию сильно несбалансированного набора данных. Если это так, то имеет смысл произвести **выборку данных с недостатком** из класса «Прочие». Мы рассмотрим понижающую передискретизацию в разделе 3.9 следующей главы.

² Чтобы экономить еще больше времени, примените кластеризацию ко всему набору непомеченных документов и помечайте вручную только те документы, которые принадлежат одному или нескольким кластерам.

скольких местах. Если мы хотим обучить модель, которая предсказывает что-то о клиентах, то можем попытаться решить задачу только для одного места или для клиентов определенного возраста.

2.2.3. Нелинейный прогресс

Прогресс проекта машинного обучения нелинеен. Ошибка предсказания обычно быстро уменьшается в начале, но затем прогресс постепенно замедляется¹. Иногда прогресс незаметен, и мы решаем включить дополнительные признаки, которые потенциально могут зависеть от внешних баз данных или баз знаний. Пока мы работаем над новым признаком или помечаем больше данных (либо поручаем эту задачу сторонней организации), никакого прогресса в качестве модели не наблюдается.

Поскольку прогресс нелинеен, необходимо позаботиться о том, чтобы владелец продукта (или клиент) понимал ограничения и риски. Тщательно регистрируйте в журнале каждое действие и отслеживайте, сколько времени оно заняло. Это поможет не только составить отчет, но и оценить сложность аналогичных будущих проектов.

2.3. ОПРЕДЕЛЕНИЕ ЦЕЛИ ПРОЕКТА МАШИННОГО ОБУЧЕНИЯ

Цель проекта машинного обучения – построить модель, которая решает или помогает решить некоторую деловую задачу. В рамках проекта модель часто рассматривается как черный ящик, описываемый структурой входных и выходных данных, а также минимально приемлемым уровнем качества (измеряемым верностью предсказания или другой **метрикой качества**).

2.3.1. Что модель может делать

Модель обычно используется как часть системы, служащая определенной цели. В частности, модель может:

- автоматизировать (например, выполнять действия от имени пользователя или запускать либо останавливать определенные операции на сервере);
- предупреждать или подсказывать (например, спрашивать пользователя, следует ли предпринимать какое-либо действие, или информировать системного администратора о том, что трафик вызывает подозрения);
- организовывать, представляя объекты в порядке, который может быть полезен пользователю (например, сортировать изображения или до-

¹ Нередко срабатывает эмпирическое правило 80/20: 80 % прогресса достигаются с использованием 20 % лучших ресурсов.

кументы в порядке сходства с запросом или в соответствии с предпочтениями пользователя);

- аннотировать (например, добавлять контекстно-зависимые аннотации к выводимой на экран информации или выделять в тексте фразы, относящиеся к задаче пользователя);
- извлекать (например, обнаруживать небольшие фрагменты релевантной информации в крупных входных данных, скажем именованные сущности в тексте: имена собственные, названия компаний или мест);
- рекомендовать (например, обнаруживать и показывать пользователю наиболее релевантные элементы коллекции, исходя из содержимого элемента или реакции пользователя на прошлые рекомендации);
- классифицировать (например, относить входные примеры к одной или нескольким предопределенным группам с уникальными названиями);
- давать количественную оценку (например, сопоставлять цену дому);
- синтезировать (например, генерировать новый текст, изображение, звук или другой объект, аналогичный объектам в коллекции);
- отвечать на прямой вопрос (например, «Описывает ли этот текст это изображение?» или «Похожи ли эти два изображения?»);
- преобразовывать входные данные (например, понижать размерность для визуализации, перефразировать длинный текст в виде краткой аннотации, переводить предложение на другой язык или дополнять изображение, применив к нему фильтр);
- обнаруживать новизну или аномалию.

Практически любую деловую задачу, решаемую с помощью машинного обучения, можно поставить в одной из перечисленных выше форм. Если это не получается, то, скорее всего, машинное обучение – не лучшее решение в вашем случае.

2.3.2. Свойства успешной модели

Успешная модель обладает следующими четырьмя свойствами:

- учитывает спецификации входных и выходных данных и требования к качеству;
- приносит пользу организации (измеряется снижением затрат, увеличением продаж или прибыли);
- помогает пользователю (измеряется продуктивностью, заинтересованностью и эмоциональным настроением);
- является строгой в научном отношении.

Научно строгая модель характеризуется предсказуемым поведением (для входных примеров, аналогичных примерам, которые использовались на этапе обучения) и воспроизводимостью. Первое свойство (предсказуемость) означает, что если входные векторы признаков получены из того же распределения значений, что и обучающие данные, то модель в среднем должна допускать тот же процент ошибок, что и на зарезервированных данных в процессе обучения. Второе свойство (воспроизводимость) означает, что модель с аналогичными свойствами может быть легко построена повторно

на основе тех же обучающих данных с применением того же алгоритма и значений гиперпараметров. Слово «легко» означает, что для реконструкции модели не требуется никакого дополнительного анализа, пометки данных или кодирования, а только вычислительные ресурсы.

Формулируя цель машинного обучения, убедитесь, что решаете правильную задачу. В качестве примера неправильно сформулированной цели представьте, что у вашего заказчика есть кошка и собака и ему нужна система, которая выпускает в дом кошку, но не выпускает собаку. Возможно, вы решите обучить модель отличать кошек от собак. Однако такая модель впустит любую кошку, а не только кошку заказчика. Или же вы можете решить, что, поскольку у клиента животные только двух видов, то следует обучить модель различать их и только их. Поскольку в данном случае модель классификации бинарная, елот будет классифицирован либо как собака, либо как кошка. Если он будет классифицирован как кошка, то попадет в дом¹.

Определить единственную цель проекта машинного обучения бывает не просто. Обычно у проекта на предприятии много заинтересованных сторон. Очевидным интересантом является владелец продукта. Допустим, что его цель – увеличить время, которое пользователь проводит на онлайн-платформе, по меньшей мере на 15 %. С другой стороны, исполнительный вице-президент хотел бы увеличить доход от рекламы на 20 %. А финансовая группа хотела бы уменьшить ежемесячный счет за пользование облаком на 10 %. При определении цели проекта машинного обучения необходимо найти правильный баланс между потенциально конфликтующими требованиями и воплотить его в выборе входных и выходных данных модели, **функции стоимости и метрике качества**.

2.4. Организация группы машинного обучения

В зависимости от предприятия есть две традиции организации группы, занимающейся МО.

2.4.1. Две традиции

Одна традиция гласит, что группа машинного обучения должна состоять из аналитиков данных, которые тесно сотрудничают с программистами. В этом случае программист не обязан обладать глубокими знаниями в области машинного обучения, но должен понимать термины, употребляемые коллегами-аналитиками.

Согласно другой традиции, все члены группы машинного обучения должны владеть навыками в области машинного обучения и разработки ПО.

У каждой традиции есть свои плюсы и минусы. Сторонники первой говорят, что каждый член группы должен быть лучшим в своем деле. Аналитик

¹ Вот почему наличие класса «Прочие» в задачах классификации почти всегда является неплохой идеей.

данных должен быть экспертом во многих технических приемах машинного обучения и обладать глубоким пониманием теории, чтобы быстро и с минимальными усилиями находить эффективное решение большинства задач. Точно так же программист должен хорошо разбираться в различных вычислительных каркасах и уметь писать эффективный и допускающий сопровождение исходный код.

Сторонники второй точки зрения говорят, что научных работников трудно включить в коллектив программистов. Научных работников больше интересует верность решений, поэтому они нередко предлагают практически не осуществимые решения, которые невозможно эффективно реализовать в производственной среде. Кроме того, поскольку ученые обычно не умеют писать эффективный, хорошо структурированный исходный код, программисту приходится его переписывать; в зависимости от проекта это может оказаться титанической работой.

2.4.2. Члены группы машинного обучения

Помимо специалистов с навыками в машинном обучении и программной инженерии, в группу МО могут входить эксперты в области инженерии данных (инженеры по данным) и эксперты по разметке данных.

Инженеры по данным – это программисты, ответственные за извлечение, преобразование и загрузку данных (Extract, Transform, Load – ETL). Эти три концептуальных шага являются частью типичного конвейера данных. Инженеры по данным используют методы ETL для создания автоматизированного конвейера, в котором исходные данные преобразуются в данные, пригодные для анализа. Они разрабатывают способы структурирования данных и их интегрирования из различных ресурсов. Они пишут запросы к данным по требованию или помещают наиболее частые запросы в быстрые API (интерфейсы прикладного программирования), чтобы аналитики и другие потребители могли легко обратиться к данным. Как правило, знакомство с машинным обучением для инженеров по данным необязательно.

В большинстве крупных компаний инженеры по данным работают отдельно от инженеров по машинному обучению в группе инженерии данных.

Эксперты по разметке данных отвечают за четыре вида деятельности:

- назначение, ручную или полуавтоматически, меток непомеченным примерам в соответствии со спецификацией, предоставленной аналитиками данных;
- создание инструментов для назначения меток данным;
- управление внешними разметчиками;
- проверка качества помеченных примеров.

Разметчик – это лицо, ответственное за назначение меток непомеченным примерам. Опять же, в крупных компаниях эксперты по разметке данных могут быть организованы в две или три разные группы: одна или две группы разметчиков (например, одна локальная, другая внешняя) и группа программистов плюс специалист по пользовательскому интерфейсу (UX), ответственный за создание инструментов разметки.

По возможности приглашайте профильных экспертов для тесного сотрудничества с научными работниками и инженерами. Привлекайте профильных экспертов к принятию решений о входных и выходных данных и признаках модели. Спрашивайте, что, по их мнению, должна предсказывать ваша модель. Тот факт, что на основе доступных данных можно предсказать некоторую величину, еще не означает, что модель будет полезна компании.

Обсуждайте с профильными экспертами, на какие особенности данных они обращают внимание при принятии решения; это поможет при конструировании признаков. Также интересуйтесь, за что именно платят клиенты и что для них является препятствием для заключения сделок; это поможет превратить деловую задачу в задачу машинного обучения.

Наконец, есть инженеры DevOps. Они тесно сотрудничают с инженерами по машинному обучению в автоматизации развертывания, загрузки, мониторинга и эпизодического или регулярного сопровождения моделей. В небольших компаниях и стартапах инженер DevOps может быть частью группы машинного обучения либо, наоборот, инженер по машинному обучению может отвечать за DevOps. В крупных компаниях занятые в проектах машинного обучения инженеры DevOps обычно приписаны к более крупной группе DevOps. Некоторые компании ввели роль MLOps, в обязанности которой входит развертывание моделей машинного обучения в производственной среде, модернизация этих моделей и создание конвейеров обработки данных с участием моделей машинного обучения.

2.5. ПРИЧИНЫ ПРОВАЛОВ ПРОЕКТОВ МАШИННОГО ОБУЧЕНИЯ

По различным оценкам, сделанным в период с 2017 по 2020 год, от 74 % до 87 % проектов в области машинного обучения и продвинутой аналитики заканчиваются провалом или не доходят до стадии эксплуатации. Причины провалов варьируются от организационных до технических. В этом разделе мы рассмотрим наиболее важные.

2.5.1. Нехватка квалифицированных кадров

По состоянию на 2020 год наука о данных, равно как инженерия машинного обучения, является относительно новой дисциплиной. До сих пор не существует стандартной методики их преподавания. Большинство организаций не знают, как нанимать специалистов по машинному обучению и как их сравнивать. Большинство доступных на рынке кадров – люди, прошедшие один или несколько онлайн-курсов и не обладающие основательным практическим опытом. Значительная часть рабочей силы обладает лишь поверхностным опытом в области машинного обучения, полученным на демонстрационных наборах данных в условиях учебных занятий. У многих нет опыта работы с полным жизненным циклом проекта машинного обуче-

ния. С другой стороны, опытные программисты, имеющиеся в организации, не обладают опытом обработки данных и построения моделей машинного обучения.

2.5.2. Отсутствие поддержки со стороны руководства

Как обсуждалось в предыдущем разделе, посвященном традициям комплектации групп машинного обучения, научные работники и программисты нередко имеют разные цели, мотивации и критерии успеха. И работают они совсем по-разному. В типичной гибкой организации группы разработки ПО работают на основе спринтов с четко сформулированными ожидаемыми результатами, когда для неопределенности остается мало места.

С другой стороны, научные работники трудятся в условиях высокой неопределенности и продвигаются вперед, ставя эксперименты. Большинство таких экспериментов не приводят к каким-либо результатам, что неопытному руководителю может показаться отсутствием прогресса. Иногда после построения и развертывания модели весь процесс приходится начинать сначала, потому что модель не приводит к ожидаемому увеличению показателя, интересующего компанию. И это тоже может восприниматься руководством как потраченные впустую время и ресурсы.

Кроме того, во многих организациях руководители, отвечающие за науку о данных и искусственный интеллект (ИИ), в особенности на уровне вице-президента, не имеют научного или хотя бы инженерного образования. Они не знают, как работает искусственный интеллект, или имеют очень поверхностное либо чрезмерно оптимистичное представление о нем, почерпнутое из научно-популярной литературы. Зачастую они полагают, что при достаточном объеме ресурсов, технических и человеческих, ИИ сможет решить любую задачу, причем быстро. Когда быстрого прогресса не наблюдается, они возлагают вину на ученых или полностью теряют интерес к ИИ как к неэффективному инструменту с трудно предсказуемыми и неопределенными результатами.

Нередко проблема заключается в неспособности научных работников донести результаты и проблемы до высшего руководства. Поскольку они говорят на разных языках и имеют несопоставимый уровень технической подготовки, даже успех, будучи плохо представлен, может быть расценен как провал.

Вот почему в успешных организациях исследователи данных являются хорошими популяризаторами, а руководители высшего звена, отвечающие за искусственный интеллект и аналитику, часто имеют техническое или научное образование.

2.5.3. Отсутствующая инфраструктура данных

Аналитики и исследователи данных работают с данными. Качество данных имеет для успеха проекта машинного обучения решающее значение. Ин-

фраструктура данных предприятия должна предлагать аналитику простые способы получения качественных данных для обучения моделей. В то же время инфраструктура должна обеспечивать доступность данных аналогичного качества после развертывания модели в производственной среде.

Однако на практике так бывает не всегда. Исследователи получают данные для обучения с помощью различных ситуативных скриптов; также используются разные скрипты и инструменты для объединения источников данных. После того как модель готова, выясняется, что, используя доступную производственную инфраструктуру, невозможно достаточно быстро (или вообще невозможно) сгенерировать входные примеры для модели. Мы подробно поговорим о хранении данных и признаков в главах 3 и 4.

2.5.4. Трудности с разметкой данных

В большинстве проектов машинного обучения аналитики используют помеченные данные. Эти данные обычно являются специализированными, поэтому разметка выполняется отдельно под каждый проект. По состоянию на 2019 год, по некоторым сообщениям¹, целых 76 % коллективов, занимающихся искусственным интеллектом и наукой о данных, размечают обучающие данные самостоятельно, тогда как 63 % создают собственные технологии автоматизации разметки и аннотирования.

Это приводит к значительным затратам времени квалифицированных исследователей данных на разметку данных и разработку инструментов. Это серьезное препятствие на пути эффективного выполнения проекта ИИ.

Некоторые компании передают разметку данных сторонним организациям на аутсорсинг. Однако без надлежащего контроля качества такие помеченные данные могут оказаться низкого качества или совершенно неверными. Предприятия в целях поддержания качества и согласованности между наборами данных должны инвестировать в формальное и стандартизированное обучение внутренних или сторонних разметчиков. Это, в свою очередь, может замедлять проекты машинного обучения. Хотя, согласно тем же отчетам, компании, которые отдают разметку данных на аутсорсинг, чаще доводят проекты машинного обучения до производственной стадии.

2.5.5. Разобщенные организации и отсутствие сотрудничества

Необходимые для проекта машинного обучения данные нередко находятся в организации в разных местах, ими владеют разные подразделения, к ним применяются различные ограничения безопасности, и представлены они в разных форматах. В разобщенных организациях люди, отвечающие за разные информационные активы, могут быть незнакомы. Отсутствие доверия

¹ What data scientists tell us about AI model training today. Alegion and Dimensional Research, 2019.

и сотрудничества приводит к трениям, когда одному подразделению требуется доступ к данным, хранящимся в другом. Кроме того, разные филиалы одной организации часто имеют свои собственные бюджеты, поэтому сотрудничество усложняется, т. к. ни одна сторона не заинтересована в том, чтобы тратить свой бюджет на помощь другой стороне.

Даже в рамках одного филиала нередко в проекте машинного обучения на разных этапах участвует несколько групп. Например, группа инженерии данных обеспечивает доступ к данным или отдельным признакам, группа исследования данных работает над моделированием, группы ETL или DevOps работают над инженерными аспектами развертывания и мониторинга, а группы автоматизации и внутреннего инструментария разрабатывают инструменты и процессы для непрерывного обновления модели. Отсутствие сотрудничества между любой парой участвующих групп может приводить к длительной приостановке проекта. Типичными причинами недоверия между группами является непонимание инженерами инструментов и подходов, используемых исследователями данных, а также отсутствие у исследователей знаний (или простая неосведомленность) о передовых методах разработки программного обеспечения и паттернах проектирования.

2.5.6. Технически невыполнимые проекты

По причине высокой стоимости многих проектов машинного обучения (из-за высокой стоимости квалифицированных кадров и инфраструктуры) некоторые организации в целях «окупаемости инвестиций» могут ставить перед собой очень амбициозные цели: полностью преобразовать организацию или продукт либо обеспечить нереалистичную рентабельность инвестиций. Это приводит к крупномасштабным проектам, предполагающим сотрудничество между многочисленными группами, подразделениями и третьими сторонами, которое требует от всех участников предельного напряжения сил.

В результате на завершение таких чрезмерно амбициозных проектов могут уходить месяцы или даже годы; некоторые ключевые игроки, включая руководителей и ведущих исследователей, могут потерять интерес к проекту или даже покинуть организацию. В конечном итоге проект может быть лишен приоритетного финансирования, а если и будет доведен до конца, то появится на рынке слишком поздно. Лучше всего, по крайней мере в начале, сосредоточиться на реализуемых проектах, предполагающих простое сотрудничество между группами, обозримость и нацеленность на простую деловую цель.

2.5.7. Нестыковка между техническими и коммерческими группами

Многие проекты машинного обучения начинаются без четкого понимания технической группой, в чем состоит коммерческая цель. Научные работники обычно формулируют задачу как классификацию или регрессию, ста-

вя в качестве цели, например, достижение высокой верности или низкой среднеквадратической ошибки. Без постоянной обратной связи со стороны коммерческой группы в части достижения деловой цели (скажем, увеличение кликабельности или удержание пользователей) исследователи нередко достигают начального уровня качества модели (в соответствии с техническим заданием), после чего не понимают, имеется ли какой-то полезный прогресс и стоит ли прилагать дополнительные усилия. В таких ситуациях проекты в конечном итоге кладутся на полку, потому что время и ресурсы были потрачены, но коммерческая группа не приняла результат.

2.6. РЕЗЮМЕ

Приступая к проекту машинного обучения, необходимо расставить приоритеты и сформировать проектную группу. Ключевыми соображениями при определении приоритетов проекта МО являются последствия и стоимость.

Последствия применения машинного обучения значительны, когда 1) машинное обучение может заменить сложную часть технического проекта или 2) велик выигрыш от получения недорогих (но, вероятно, несовершенных) предсказаний.

На стоимость проекта машинного обучения сильно влияют три фактора: 1) сложность задачи, 2) стоимость данных и 3) необходимое качество модели.

Стандартного метода оценивания сложности проекта машинного обучения не существует – разве что сравнение с другими проектами, выполненными в организации или описанными в литературе. Есть несколько существенных неизвестных, которые почти невозможно угадать: достигим ли требуемый уровень качества модели на практике, сколько данных потребуется для достижения этого уровня, сколько и каких признаков необходимо, насколько большой должна быть модель, сколько времени потребуется для выполнения одного эксперимента и сколько экспериментов необходимо для достижения желаемого уровня качества.

Гипотеза будет более обоснованной, если упростить задачу.

Прогресс проекта машинного обучения нелинеен. Ошибка обычно быстро уменьшается в начале, но затем прогресс замедляется. Поэтому необходимо позаботиться о том, чтобы владелец продукта (или клиент) понимал ограничения и риски. Тщательно регистрируйте в журнале каждое действие и отслеживайте, сколько времени оно заняло. Это поможет не только составить отчет, но и оценить сложность аналогичных будущих проектов.

Цель проекта машинного обучения – построить модель, которая решает деловую задачу. В частности, модель может использоваться в рамках более широкой системы для автоматизации, оповещения, организации, аннотирования, извлечения, генерирования рекомендаций, классификации, количественной оценки, синтеза, ответа на прямые вопросы, преобразования входных данных и обнаружения новизны или аномалии. Если не получается сформулировать цель машинного обучения в одной из этих форм, то, скорее всего, машинное обучение – не лучшее решение.

Успешная модель:

- учитывает спецификации входных и выходных данных и требования к качеству;
- приносит пользу организации и пользователю;
- является строгой в научном отношении.

Существует две традиции организации групп машинного обучения. Одна утверждает, что группа машинного обучения должна состоять из аналитиков данных, которые тесно сотрудничают с программистами. В этом случае программист не обязан обладать глубокими знаниями в области машинного обучения, но должен понимать термины, употребляемые коллегами-аналитиками. Согласно другой традиции, все члены группы машинного обучения должны владеть навыками в области машинного обучения и разработки ПО.

Помимо специалистов с навыками в области машинного обучения и разработки ПО, в группу машинного обучения могут входить эксперты по разметке данных и по инженерии данных. Инженеры DevOps тесно сотрудничают с инженерами по машинному обучению для автоматизации развертывания, загрузки, мониторинга и эпизодического или регулярного сопровождения моделей.

Проекты машинного обучения могут терпеть неудачу по многим причинам, и большинство из них действительно оказываются провальными. Типичные причины провала таковы:

- нехватка квалифицированных кадров;
- отсутствие поддержки со стороны руководства;
- отсутствие инфраструктуры данных;
- трудности с разметкой данных;
- разобщенность организации и отсутствие сотрудничества;
- техническая невыполнимость проекта;
- нестыковка между техническими и коммерческими группами.

Глава 3

Сбор и подготовка данных

Прежде чем приступить к какой-либо деятельности, связанной с машинным обучением, аналитик должен собрать и подготовить данные. Доступные аналитику данные не всегда «правильные» и не всегда представлены в форме, подходящей для алгоритма машинного обучения. В этой главе нас будет интересовать второй этап жизненного цикла проекта машинного обучения, как показано ниже:

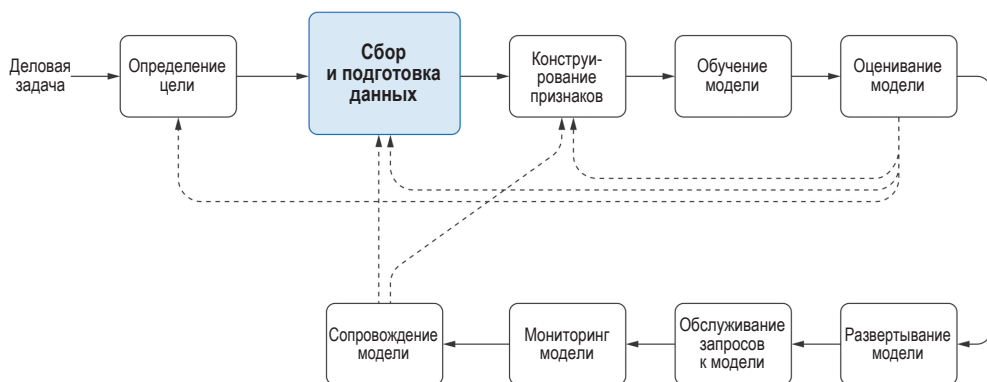


Рис. 3.1 ❖ Жизненный цикл проекта машинного обучения

В частности, мы поговорим о свойствах высококачественных данных, типичных проблемах набора данных и способах подготовки и хранения данных для машинного обучения.

3.1. Вопросы к данным

Итак, мы сформулировали цель машинного обучения, определили входные и выходные данные модели и критерии успеха. Теперь можно начать сбор данных, необходимых для обучения модели. Но прежде необходимо ответить на несколько вопросов.

3.1.1. Доступны ли данные?

Существуют ли необходимые данные? Если да, то доступны ли они (физически, по контракту, этически или с точки зрения стоимости)? Если вы покупаете или используете чужие источники данных, подумали ли вы о том, на каких условиях эти данные можно использовать или передавать в общее пользование? Нужно ли договариваться о новом лицензионном соглашении с владельцем?

Если данные доступны, то защищены ли они авторским правом или другими правовыми нормами? Если да, то установили ли вы, кому принадлежат авторские права на эти данные? Возможно ли совместное авторское право?

Являются ли данные конфиденциальными (например, касаются проектов, клиентов или партнеров вашей организации либо засекречены правительством)? Возможны ли потенциальные проблемы с конфиденциальностью? Если да, то обсуждали ли вы коллективное использование данных с респондентами, от которых получали данные? Можете ли вы обеспечить сохранность персональных данных в течение длительного времени, чтобы их можно было использовать в будущем?

Нужно ли распространять данные вместе с моделью? Если да, то нужно ли получать письменное согласие от владельцев или респондентов?

Нужно ли обезличивать данные¹, например путем удаления **информации, позволяющей установить личность** (personally identifiable information – PII), во время анализа или при подготовке к распространению?

Даже если получить нужные данные физически возможно, не работайте с ними до тех пор, пока не будут решены все вышеперечисленные вопросы.

3.1.2. Насколько велик объем данных?

Обязательно нужно получить точный ответ на вопрос, достаточно ли данных. Но, как мы уже выяснили, обычно объем данных, необходимых для достижения цели, неизвестен, особенно если предъявляются строгие требования к минимальному качеству модели.

Если есть сомнения в доступности достаточного объема данных немедленно, то следует выяснить, с какой частотой генерируются новые данные. В некоторых проектах можно начать с того, что доступно на начальной стадии, и постепенно накапливать новые данные, пока идет работа над конструированием признаков, моделированием и решением других технических задач. Данные могут поступать естественным путем, как результат какого-либо

¹ Наглядным примером является политика распространения контента Twitter. Она ограничивает распространение любой информации о твитах, кроме идентификаторов твитов и идентификаторов пользователей. Twitter хочет, чтобы посредством API Twitter аналитики получали только последние данные. Одним из возможных объяснений такого ограничения является то, что некоторые пользователи, возможно, захотят удалить конкретный твит, потому что передумали или сочли его слишком спорным. Если этот твит уже был извлечен из хранилища и размещен в публичном пространстве, то это может сделать данного пользователя уязвимым.

наблюдаемого или измеряемого процесса, либо предоставляться экспертами по разметке данных или сторонним поставщиком данных.

Оцените, сколько времени может занять выполнение проекта. Будет ли за это время собран достаточно большой набор данных¹? Ответ должен быть основан на опыте работы над аналогичными проектами или на результатах, описанных в литературе.

Практическим способом оценить достаточность собранных данных является построение **кривых обучения**. Именно, постройте кривые оценок на обучающем и контрольном наборах при различном числе обучающих примеров, как показано на рис. 3.2.

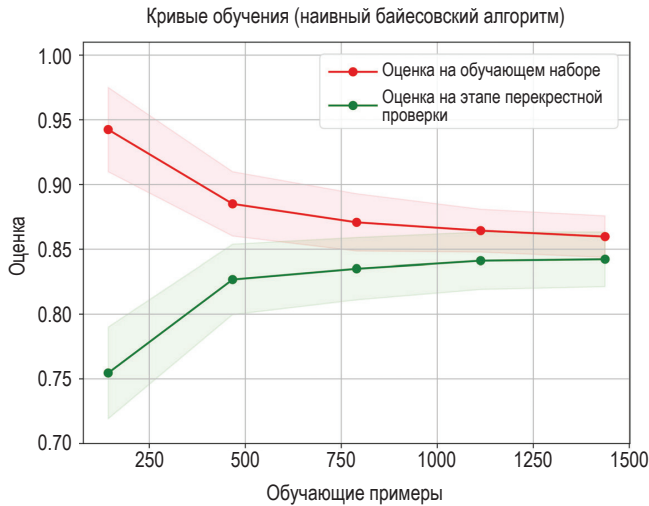


Рис. 3.2 ❖ Кривые обучения для наивного байесовского алгоритма применительно к стандартному набору данных цифр из библиотеки scikit-learn

Глядя на кривые обучения, вы увидите, что качество модели будет выходить на плато после определенного числа обучающих примеров. Дальнейшее увеличение их количества не приносит ощутимой отдачи.

Если вы замечаете, что качество алгоритма обучения достигло насыщения, то, скорее всего, сбор дополнительных данных не поможет улучшить модель. Я говорю «скорее всего», потому что возможны два других объяснения:

- отсутствуют достаточно информативные признаки, которые алгоритм обучения мог бы использовать для построения более качественной модели;
- использованный алгоритм обучения не способен обучить достаточно сложную модель с применением имеющихся данных.

¹ Делая оценку, не забывайте, что для проверки качества модели на не предъявлявшихся ранее примерах понадобятся не только обучающие, но и зарезервированные данные. Зарезервированных данных тоже должно быть достаточно для обеспечения статистически надежных оценок качества модели.

В первом случае можно рассмотреть вопрос о конструировании дополнительных признаков, изобретательно скомбинировав существующие признаки или используя информацию из сторонних источников, например справочных таблиц и географических справочников. Мы рассмотрим методы синтеза признаков в разделе 4.6 главы 4.

Во втором случае можно было бы воспользоваться ансамблевым обучением или обучить глубокую нейронную сеть. Однако глубокие нейронные сети обычно требуют большего объема обучающих данных по сравнению с алгоритмами поверхностного обучения.

Иногда используют эмпирические правила для оценивания количества обучающих примеров, необходимых для решения задачи. Обычно применяются коэффициенты масштабирования к:

- числу признаков, либо
- числу классов, либо
- к числу обучаемых параметров модели.

Такие эмпирические правила нередко работают, но зависят от предметной области. Каждый аналитик вносит поправки, опираясь на собственный опыт. Конечно, со временем вы найдете свои «волшебные» масштабные коэффициенты, а пока можете ориентироваться на цифры, которые чаще всего приводятся в онлайн-источниках:

- в 10 раз увеличить количество признаков (при этом размер обучающего набора часто оказывается завышен, но в качестве верхней границы эта рекомендация хорошо работает);
- в 100 или 1000 раз увеличить количество классов (при этом размер часто занижается);
- в 10 раз увеличить количество обучаемых параметров (обычно применяется к нейронным сетям).

Имейте в виду, что сам факт доступа к большому объему данных еще не означает, что все их нужно использовать. Сравнительно небольшая выборка может дать хорошие результаты на практике и ускорить поиск более качественной модели. Однако важно убедиться, что выборка репрезентативна для всего набора данных. Такие стратегии, как **стратифицированная** или **систематическая выборка**, могут улучшить результаты. Мы рассмотрим стратегии выборки в разделе 3.10.

3.1.3. Пригодны ли данные для использования?

Качество данных является одним из важнейших факторов, влияющих на качество модели. Допустим, требуется обучить модель, которая предсказывает пол человека по его имени. Мы можем получить набор данных о людях, содержащий гендерную информацию. Однако если использовать этот набор данных вслепую, то, как бы мы ни старались, качество модели на новых данных останется низким. В чем же причина?

Возможный ответ состоит в том, что информация о поле была не первичной, а полученной от статистического классификатора низкого качества. В этом случае качество модели не может быть лучше качества классификатора.

Если набор данных представлен в виде электронной таблицы, то прежде всего нужно проверить, аккуратны ли данные. Как было сказано во введении, используемый для машинного обучения набор данных должен быть аккуратным. Если для ваших данных это не так, то следует преобразовать их в аккуратные, применив конструирование признаков.

В аккуратном наборе данных некоторые значения могут **отсутствовать**. Для их восполнения рассмотрите возможность применения методов **подстановки данных**. Мы обсудим несколько таких методов в разделе 3.7.

В наборах данных, подготовленных людьми, вместо отсутствующих значений часто задаются **магические числа** типа 9999 или -1. Такие ситуации должны быть обнаружены на этапе визуального анализа данных, после чего магические числа должны быть заменены с помощью подстановки данных.

Необходимо также проверить наличие **дубликатов** в наборе данных. Обычно дубликаты удаляются, если только не были добавлены специально, чтобы сбалансировать **несбалансированную задачу**. Мы рассмотрим эту проблему и методы ее решения в разделе 3.9.

Срок действия данных может **истечь**, или они могут значительно устареть. Например, предположим, что требуется обучить модель, которая распознает отклонения в поведении сложного электронного устройства, например принтера. У нас имеются измерения, произведенные во время нормальной и аномальной работы принтера. Однако они были сделаны для принтеров предыдущего поколения, а с тех пор было внедрено несколько значительных усовершенствований. Модель, обученная с использованием неактуальных данных о более старом поколении принтеров, может работать хуже при развертывании на принтерах нового поколения.

Наконец, данные могут быть **неполными** или **нерепрезентативными**. Например, набор фотографий животных может содержать снимки, сделанные только летом или в определенной географической области. Набор данных о пешеходах для беспилотных автомобилей может быть создан инженерами, игравшими роль пешеходов; в таком наборе данных чаще всего будут представлены только молодые мужчины, тогда как дети, женщины и пожилые люди будут встречаться редко или вообще отсутствовать.

Научно-исследовательский отдел компании, работающей над распознаванием выражений лиц, может быть расположен в стране, где преобладает белое население, поэтому в наборе данных будут присутствовать в основном лица белых мужчин и женщин, а чернокожие и азиаты будут представлены недостаточно. Инженеры, разрабатывающие модель распознавания позы для камеры, возможно, создавали обучающий набор данных, фотографируя людей в помещении, тогда как потребители обычно используют камеру на открытом воздухе.

На практике часто бывает, что для подачи на вход модели данные необходимо подвергнуть **предобработке**; отсюда важность визуального анализа набора данных перед началом моделирования. Допустим, мы работаем над задачей предсказания темы новостных статей. Скорее всего, данные будут собираться с новостных сайтов. А даты скачивания, вероятно, будут сохранены в том же документе, что и текст новостной статьи. Допустим также, что инженер по данным решил в цикле перебирать встречающиеся на сайтах

новости и обрабатывать по одной теме на каждой итерации. То есть в понедельник были обработаны новости, относящиеся к искусству, во вторник – к спорту, в среду – к технологиям и т. д.

Если не подвергнуть такие данные предобработке, удалив даты, то модель может обучиться корреляции даты и темы, поэтому на практике будет бесполезна.

3.1.4. Понятны ли данные?

Как показано на примере предсказания пола, крайне важно понимать, откуда взялся каждый атрибут в наборе данных. Не менее важно понимать, что именно представляет каждый атрибут. Одна из частых проблем, наблюдаемых на практике, заключается в том, что переменная, которую аналитик пытается предсказать, находится среди признаков в векторе признаков. Как так?

Допустим, мы работаем над задачей предсказания цены дома по его атрибутам: число спален, площадь, местоположение, год постройки и т. д. Атрибуты каждого дома были предоставлены нам клиентом, крупной онлайн-платформой по продаже недвижимости. Данные представлены в виде электронной таблицы Excel. Не тратя слишком много времени на анализ каждого столбца, мы удаляем из атрибутов только цену сделки и используем это значение в качестве целевого показателя, который хотим научиться предсказывать. Очень быстро приходит понимание, что модель почти идеальна: она предсказывает цену сделки с верностью, близкой к 100 %. Мы передаем модель клиенту, он внедряет ее в производство, и тесты показывают, что в большинстве случаев модель врёт. Что случилось?

То, что произошло, называется **просачиванием данных** (или **просачиванием цели**). После более тщательного анализа набора данных мы понимаем, что в одном из столбцов электронной таблицы указана комиссия риелтора. Конечно же, модель легко обучилась идеально преобразовывать этот атрибут в цену дома. Однако в производственной среде эта информация недоступна до момента продажи дома, поскольку комиссия зависит от продажной цены. В разделе 3.2.8 мы рассмотрим проблему просачивания данных подробнее.

3.1.5. Надежны ли данные?

Надежность набора данных зависит от того, как данные собирались. Можно ли доверять меткам? Если данные порождены работниками с помощью «Механического турка» (так называемыми «туркерами»)¹, то их надежность

¹ Механический турок (Mechanical Turk) – виртуальный рынок труда, созданный компанией Amazon в 2005 году. Эта платформа дает работодателям возможность предлагать разнообразные, преимущественно простые, задания широкому кругу пользователей. Механический турок может использоваться для самых разных задач: категоризация, верификация данных, расстановка тегов, транскрибирование или перевод. – *Прим. перев.*

может оказаться очень низкой. Иногда метки, назначенные векторам признаков, получены большинством голосов (или средним арифметическим) нескольких пользователей. В таком случае данные можно считать более надежными. Однако лучше проводить дополнительный контроль качества на небольшой случайной выборке из набора данных.

С другой стороны, если данные представляют измерения, произведенные измерительными приборами, то информацию о точности каждого измерения можно найти в технической документации.

На надежность меток также может влиять **отсроченная** или **косвенная** природа метки. Метка считается отсроченной, если вектор признаков, которому была назначена метка, представляет событие, произошедшее значительно раньше времени наблюдения метки.

Конкретно возьмем задачу **предсказания оттока клиентов**. Здесь вектор признаков описывает клиента, и мы хотим предсказать, уйдет ли клиент в какой-то момент в будущем (обычно спустя время от шести месяцев до года). Вектор признаков содержит то, что мы знаем о клиенте сейчас, но метка («ушел» либо «остался») будет назначена в будущем. Это важное свойство, потому что между настоящим и будущим может произойти много событий, которые не отражены в векторе признаков, но влияют на решение клиента остаться или уйти. Следовательно, отсроченные метки делают данные менее надежными.

Надежность также зависит от того, является ли метка прямой или косвенной, хотя тут, конечно, важно, что мы хотим предсказать. Пусть, например, наша цель – предсказать, будет ли посетителю сайта интересна некоторая веб-страница. Мы могли бы получить набор данных, содержащий информацию о пользователях, веб-страницах и метках «заинтересован»/«не заинтересован», отражающих, был ли конкретный пользователь заинтересован в определенной веб-странице или нет. Прямая метка действительно указывала бы на интерес, а косвенная метка могла бы предполагать *некоторый* интерес. Например, если пользователь нажал кнопку «Нравится», то мы имеем прямой индикатор интереса. Но если пользователь только перешел по ссылке, то это является лишь косвенным выражением заинтересованности. Пользователь мог перейти по ошибке или потому, что текст ссылки был «заманухой», – мы не можем знать наверняка. Если метка является косвенной, то данные следует считать менее надежными. Впрочем, они менее надежны для предсказания интереса к странице, но вполне надежны для предсказания переходов.

Еще одним источником ненадежности данных являются петли обратной связи. **Петля обратной связи** – это свойство системы, позволяющее получать данные для обучения модели с помощью самой модели. Снова рассмотрим задачу предсказания того, понравится ли контент пользователю сайта, и предположим, что имеются только косвенные метки – клики. Если модель уже развернута на сайте и пользователи переходят по рекомендованным моделью ссылкам, то это означает, что новые данные косвенно отражают не только интерес пользователей к контенту, но и то, насколько интенсивно модель рекомендовала этот контент. Если модель решит, что конкретная ссылка достаточно важна, чтобы рекомендовать ее многим пользователям,

то, скорее всего, больше пользователей перейдут по этой ссылке, в особенности если рекомендация была неоднократно повторена в течение нескольких дней или недель.

3.2. Типичные проблемы с данными

Как мы только что видели, данные, с которыми вам предстоит работать, могут быть подвержены проблемам. В этом разделе мы перечислим наиболее важные из них и расскажем, как их можно смягчить.

3.2.1. Высокая стоимость

Получение непомеченных данных может обходиться дорого; однако разметка данных – самая дорогостоящая работа, в особенности если она выполняется вручную.

Получение непомеченных данных становится дорогостоящим, когда их необходимо собирать специально для конкретной задачи. Допустим, наша цель – узнать, где в городе расположены разные торговые предприятия. Лучшее всего было бы купить эти данные у государственного учреждения. Однако по разным причинам это может быть сложно или даже невозможно: государственная база данных может быть неполной или устаревшей. Для получения актуальных данных можно выпустить оснащенные камерами автомобили на улицы города и поручить им фотосъемку всех зданий.

Понятно, что такая затея стоит недешево. Собрать фотографии зданий недостаточно. Нам еще нужен тип торгового предприятия в каждом здании. То есть нужны данные, помеченные как «кофейня», «банк», «бакалея», «аптека», «заправочная станция» и т. д. Метки должны назначаться вручную, и кому-то за выполнение этой работы придется дорого заплатить. Кстати, у Google есть хитроумный метод передачи разметки на аутсорсинг случайным людям с помощью бесплатной службы reCAPTCHA. Благодаря службе reCAPTCHA компания Google решает две проблемы: сокращение спама в вебе и получение дешевых помеченных данных.

На рис. 3.3 показано, что нужно сделать для назначения меток одному изображению. Цель здесь состоит в том, чтобы сегментировать изображение, назначая каждому пикселю метки: «тяжелый грузовик», «легковой автомобиль или легкий грузовик», «лодка», «здание», «контейнер», «иное». Разметка изображения на рис. 3.3 заняла у меня около 30 минут. Если бы типов было больше, например «мотоцикл», «дерево», «дорога», то это заняло бы больше времени, а стоимость разметки была бы выше.

В хорошо продуманных инструментах разметки минимизировано использование мыши (в т. ч. для открытия меню), по максимуму используются горячие клавиши, поэтому затраты снижаются за счет увеличения скорости разметки данных.

По возможности процесс принятия решений следует сводить к выбору ответа да или нет. Вместо того чтобы просить найти все цены в данном тексте,

извлеките все числа, а затем поочередно выводите их на экран, спрашивая, является ли это число ценой, как показано на рис. 3.4. Если разметчик нажмет «Не уверен», то пример можно сохранить для последующего анализа или просто не использовать такие примеры для обучения модели.

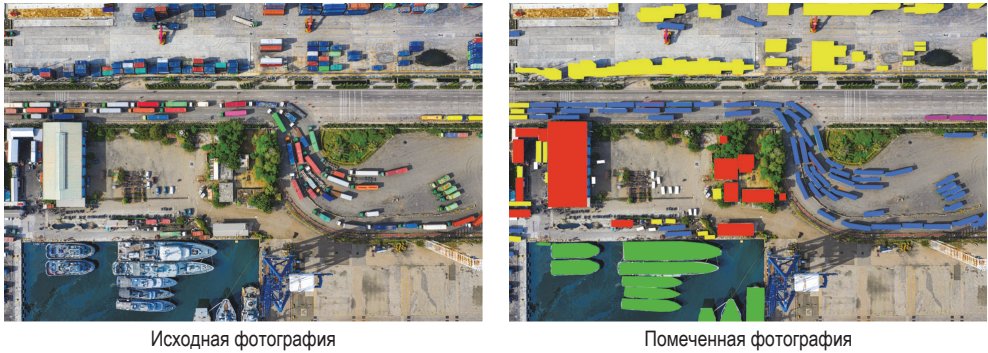


Рис. 3.3 ❖ Непомеченный и помеченный аэрофотоснимок.
Фото предоставлено Томом Фиском



Рис. 3.4 ❖ Пример простого интерфейса разметки

Еще один прием, позволяющий ускорить разметку, – **зашумленная предварительная разметка**, т. е. предварительная пометка примера с использованием наилучшей текущей модели. В этом сценарии мы начинаем с пометки определенного числа примеров «с нуля» (то есть без использования какой-либо поддержки). Затем с помощью этого начального набора помеченных примеров создается первая модель, которая работает более-менее приемлемо. После этого для пометки новых примеров используется текущая модель вместо разметчика-человека¹.

Спросите, правильна ли автоматически назначенная метка. Если разметчик отвечает «Да», то сохраните этот пример как обычно. Если же он отвечает «Нет», то попросите пометить этот пример вручную. Технологический про-

¹ Вот почему этот подход называется «зашумленной» предварительной разметкой: метки, назначаемые примерам с использованием неоптимальной модели, не всегда верны и требуют проверки человеком.

цесс показан на рис. 3.5. Цель хорошо организованного процесса – максимально ускорить разметку. Интерес разметчика к процессу также является ключевым фактором. Показывайте, сколько меток было добавлено, а также качество текущей наилучшей модели. Это стимулирует разметчика, поскольку у него появляется ясная цель.

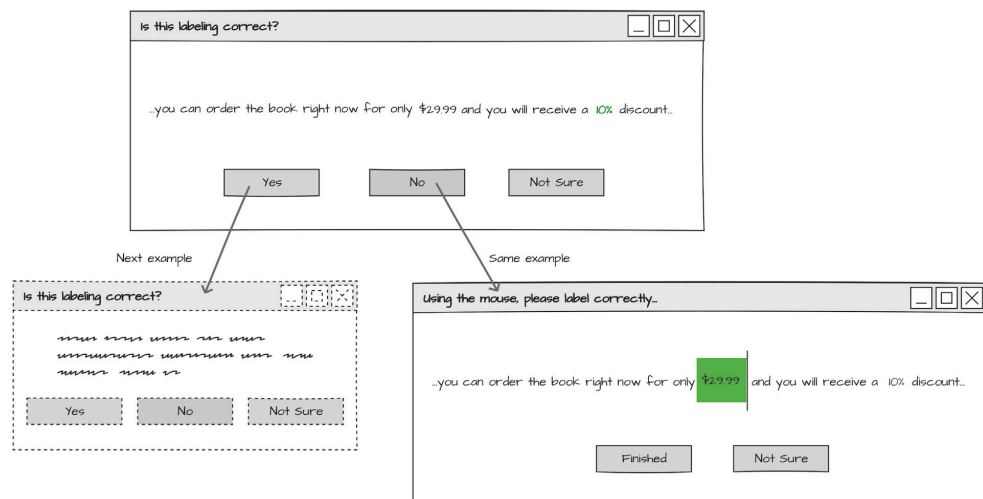


Рис. 3.5 ❖ Пример технологического процесса зашумленной предварительной разметки

3.2.2. Плохое качество

Помните, что качество данных является одним из важнейших факторов, влияющих на качество модели. Не устаю это подчеркивать.

У качества данных есть два аспекта: качество первичных данных и качество разметки.

Для первичных данных характерно несколько хорошо известных проблем: зашумленность, смещение, низкая предсказательная способность, устаревшие примеры, выбросы и просачивание.

3.2.3. Зашумленность

Зашумленность данных – это повреждение примеров. Изображения могут быть размытыми или неполными. Форматирование текста может быть нарушено, что приводит к слиянию или разбивке некоторых слов. Аудиоданные могут содержать фоновый шум. Ответы на опрос могут быть неполными, некоторые атрибуты, например возраст или пол респондента, могут отсутствовать. Шум часто является случайным процессом, который искажает каждый пример независимо от остальных.

Если в аккуратных данных пропущены атрибуты, то их можно восполнить, применив методы **подстановки данных**, которые мы рассмотрим в разделе 3.7.1.

Восстановить резкость размытых изображений позволяют специальные алгоритмы, хотя при необходимости этому можно обучить модели глубокого машинного обучения, например нейронные сети. То же самое относится к зашумленности аудиоданных: шум можно устранить алгоритмически.

Зашумленность является серьезной проблемой, когда набор данных относительно невелик (тысячи примеров или меньше), поскольку наличие шума может стать причиной **переобучения**: алгоритм может обучиться моделировать шум, содержащийся в обучающих данных, что нежелательно. С другой стороны, в контексте больших данных шум, случайно применяемый к каждому примеру независимо от всех остальных, обычно «усредняется» по всем примерам. И в этом случае результатом шума может стать регуляризация, поскольку он не дает алгоритму обучения слишком сильно полагаться на малое подмножество входных признаков¹.

3.2.4. Смещение

Смещение данных – это несогласованность с явлением, которое данные представляют. Она может возникать по ряду причин (которые не являются взаимоисключающими).

Типы смещения

Смещение отбора отдает предпочтение источникам данных, которые легко доступны, удобны и (или) экономически эффективны. Например, вы хотите узнать мнение читателей о своей новой книге. Для этого вы рассылаете несколько начальных глав читателям предыдущей книги. Очень вероятно, что этой избранной группе ваша новая книга понравится. Однако эта информация мало что говорит о читателе вообще.

Реальным примером смещения отбора является изображение, сгенерированное с помощью алгоритма повышающей передискретизации фотографий Photo Upsampling via Latent Space Exploration (**PULSE**), в котором используется нейросетевая модель для увеличения разрешающей способности. Во время его тестирования пользователи интернета обнаружили, что передискретизированное изображение чернокожего в некоторых случаях может представлять белого человека, как показано на фотографии Барака Обамы на рис. 3.6.

Этот пример показывает, что нельзя предполагать, что модель машинного обучения верна просто потому, что алгоритмы машинного обучения беспристрастны, а обученные модели основаны на данных. Если в данных имеется смещение, то оно, скорее всего, будет отражено и в модели.

¹ Это, кстати, является причиной повышения качества за счет **прореживания**, которое в **глубоком обучении** является методом регуляризации.

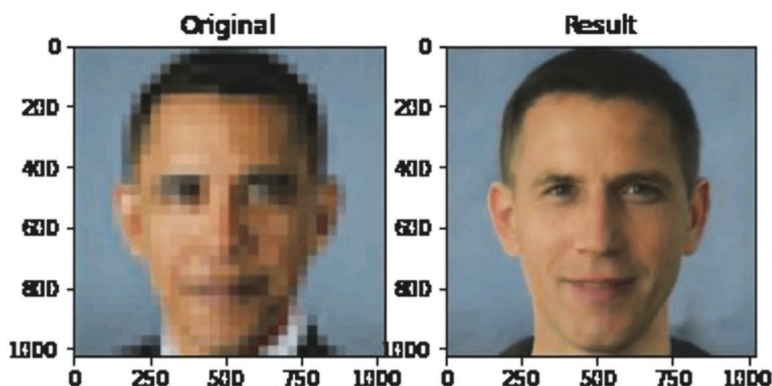


Рис. 3.6 ❖ Влияние смещения отбора на обученную модель
(Твиттер / @Chicken3gg)

Смещение самоотбора – это форма смещения отбора, когда данные поступают из «добровольно» предложивших их источников. Такому смещению подвержены данные большинства опросов. Пусть, например, мы хотим обучить модель предсказывать поведение успешных предпринимателей. Сначала мы задаем предпринимателям вопрос, успешны они или нет. И сохраняем данные, только если респондент заявил, что успешен. Проблема в том, что у действительно успешных предпринимателей, скорее всего, нет времени отвечать на наши вопросы, а те, кто объявляет себя успешными, могут ошибаться на свой счет.

Другой пример. Пусть мы хотим обучить модель предсказывать, понравится ли книга читателям. Можно воспользоваться оценками, которые пользователи ставили похожим книгам в прошлом. Однако хорошо известно, что недовольные пользователи часто ставят несоразмерно низкие оценки. Данные будут смещены в сторону большого количества очень низких оценок по сравнению с числом оценок в среднем диапазоне (рис. 3.7). Смещение усугубляется тем фактом, что мы склонны ставить оценку, только если наш опыт оказался очень положительным или очень отрицательным.



Рис. 3.7 ❖ Распределение оценок,
поставленных читателям популярной книге по ИИ на сайте Amazon

Смещение из-за отсутствующей переменной возникает, когда среди выделенных из данных признаков нет нужного для верного предсказания. Предположим, к примеру, что мы разрабатываем модель оттока клиентов и хотим предсказывать, аннулирует ли клиент подписку в течение шести месяцев. Мы обучили модель, она достаточно верна, но спустя несколько недель после развертывания мы начинаем наблюдать много неожиданных ложноотрицательных результатов. Мы провели расследование снижения качества модели и обнаружили, что появился конкурент, предлагающий аналогичную услугу за меньшую цену. Мы не учли этот признак в модели, поэтому не оказалось важной информации, необходимой для верного предсказания.

Смещению из-за спонсорства или финансирования подвержены данные, предоставленные спонсируемым агентством. Например, предположим, что известная компания по производству видеоигр спонсирует новостное агентство, которое поставяет новости об игровой индустрии. Пытаясь дать предсказания, касающиеся индустрии видеоигр, мы можем включить в данные новости, предоставляемые этим спонсируемым агентством. Однако спонсируемое агентство склонно не афишировать плохие новости о своем спонсоре и преувеличивать его достижения.

Смещение выборки (или сдвиг распределения) возникает, когда распределение примеров, используемых для обучения, не отражает распределение входных данных, получаемых моделью на этапе эксплуатации. Такой тип смещения часто наблюдается на практике. Пусть, например, мы разрабатываем систему, которая классифицирует документы по нескольким сотням тем. Мы создали набор документов, в котором каждая тема представлена одинаковым количеством документов. По завершении разработки модель давала ошибку 5 %. Но вскоре после развертывания обнаруживается, что неправильно классифицируется примерно 30 % документов. Что же случилось?

Одна из возможных причин – смещение выборки: на одну или две часто встречающиеся темы может приходиться 80 % всех входов. Если модель плохо работает для таких частых тем, то развернутая система будет давать больше ошибок, чем ожидалось.

Смещение из-за предубеждения или стереотипов нередко наблюдается, когда данные получены из исторических источников, например книг и фотоархивов, или из онлайн-источников типа социальных сетей, сетевых форумов и комментариев к онлайн-публикациям. Например, при использовании фотоархива для обучения модели, различающей мужчин и женщин, может оказаться, что мужчины чаще встречаются за работой или на улице, а женщины – внутри помещения. Модель, обученная на таких смещенных данных, может испытывать трудности при распознавании женщин на улице или мужчин в доме.

Знаменитый пример смещения такого типа – поиск ассоциаций со словами с использованием **погружений слов**, обученных алгоритмом типа **word2vec**. Такая модель предсказывает, что $\text{king} - \text{man} + \text{woman} \approx \text{queen}$ ¹ и в то же время что $\text{programmer} - \text{man} + \text{woman} \approx \text{homemaker}$ ².

¹ Король – мужчина + женщина ≈ королева. – Прим. перев.

² Программист – мужчина + женщина ≈ домохозяйка. – Прим. перев.

Систематическая ошибка обычно возникает, когда для измерений или наблюдений используется какое-то устройство. В результате модель машинного обучения дает неоптимальные предсказания после развертывания в производственной среде.

Пусть, например, обучающие данные собираются с помощью камеры, настроенной так, что баланс белого дает немного желтоватый цвет. Но на этапе эксплуатации использовалась высококачественная камера, которая «видит» белый цвет как белый. Поскольку модель была обучена на фотографиях худшего качества, ее предсказания для высококачественных изображений будут неоптимальны.

Не следует путать систематическую ошибку с зашумленными данными. Шум – это результат случайного процесса, искажающего данные. При наличии достаточно большого набора данных шум перестает быть проблемой, потому что усредняется. С другой стороны, если измерения всегда смещены в одном направлении, то это ухудшает обучающие данные и в конечном итоге приводит к модели низкого качества.

Необъективность экспериментатора – это тенденция искать, интерпретировать, отдавать предпочтение или отбирать информацию так, что она подтверждает априорные убеждения или предположения автора. Применительно к машинному обучению необъективность экспериментатора часто возникает, когда все примеры в наборе данных получены в результате опроса, проведенного конкретным лицом.

Обычно опросный лист содержит несколько вопросов. Их форма может оказать существенное влияние на ответы. Вот простейший пример: «Какой сорт пиццы вы предпочитаете: пепперони, мясную или вегетарианскую?» Тут нет возможности выбрать вариант, отличный от предлагаемых, или хотя бы ответить «Другой». Или же вопрос может быть сформулирован тенденциозно. Например, вместо того чтобы спросить «Вы принимаете участие в переработке мусора?», необъективный аналитик может задать вопрос: «Вы уклоняетесь от переработки мусора?» В первом респондент скорее даст честный ответ, чем во втором.

Необъективность экспериментатора может также возникать, если аналитик заранее проинструктирован подкрепить определенный вывод (например, в пользу вести «бизнес как обычно»). В таком случае аналитик может исключить некоторые переменные как ненадежные или зашумленные.

Смещение пометки возникает, когда процесс или человек, назначающий метки непометенным примерам, необъективен. Например, если вы просите нескольких разметчиков назначить тему документу, предварительно прочитав его, то одни разметчики прочитают документ целиком и присвоят метки осознанно, а другие просмотрят текст «по диагонали», выхватят несколько ключевых фраз и выберут тему, лучше всего отражающую эти фразы. Поскольку каждый человек обращает больше внимания на ключевые фразы из близких себе предметных областей и меньше на другие, то метки, назначенные разметчиками, не прочитавшими текст внимательно, будут смещены.

С другой стороны, некоторые разметчики могут отдать предпочтение документам на интересные им темы. Такой разметчик будет пропускать не-

интересные документы, которые, следовательно, будут недостаточно представлены в данных.

Как избежать смещения

Обычно мы не можем точно знать, какие смещения присутствуют в наборе данных. Но даже если бы знали, избежать их – непростая задача. Главное – быть готовым.

Полезно всегда задаваться вопросами: кто создавал данные, какие у него были побуждения и критерии качества и, что еще важнее, как и почему создавались данные. Если данные являются результатом какого-то исследования, то поинтересуйтесь методом исследования и убедитесь, что не было внесено никаких из вышеописанных смещений.

Смещение отбора можно предотвратить, систематически задавая вопрос о причине выбора конкретного источника данных. Если это простота или низкая стоимость, то отнеситесь настороженно. Вспомните пример, когда требуется узнать, подпишется ли клиент на вашу новую услугу. Обучать модель только на данных о текущих клиентах – скорее всего, неудачная идея, потому что существующие клиенты более лояльны к вашей торговой марке, чем случайно выбранный потенциальный клиент. Следовательно, оценка качества модели будет чрезмерно оптимистичной.

Смещения самоотбора полностью избежать невозможно. Обычно оно возникает, когда данные получены в результате опроса; сам факт согласия респондента ответить на вопросы уже является проявлением смещения самоотбора. Чем длиннее опросный лист, тем меньше шансов, что респондент будет отвечать на все вопросы сосредоточенно. Поэтому старайтесь делать опрос покороче и предлагайте стимулы давать качественные ответы.

Заранее отбирайте респондентов, чтобы минимизировать самоотбор. Не спрашивайте предпринимателей, считают ли они себя успешными. Вместо этого составьте список на основе характеристик, почерпнутых от экспертов или из публикаций, и обращайтесь только к лицам из этого списка.

Избежать **смещения из-за отсутствующей переменной** трудно, потому что, как говорится, «чего не знаем, того не знаем». Возможный подход – использовать всю доступную информацию, т. е. включить столько признаков, сколько возможно, даже если они кажутся ненужными. В результате вектор признаков может получиться очень широким (имеющим много измерений) и разреженным (значения большинства измерений нулевые). Тем не менее если хорошо настроить регуляризацию, то модель сама «решит», какие признаки важны, а какие нет.

С другой стороны, мы можем заподозрить, что некоторая переменная важна для верности предсказаний, и если не включить ее в модель, то появится смещение из-за отсутствующей переменной. Предположим, что получить эти данные проблематично. Тогда попробуйте воспользоваться замещающей переменной. Например, если мы хотим обучить модель, которая предсказывает цену подержанного автомобиля, но не можем узнать возраст автомобиля, то можно заменить его сроком, в течение которого автомобиль принадлежал текущему владельцу. Это будет заместитель возраста транспортного средства.

Смещение из-за спонсорства можно уменьшить, внимательно исследовав источник данных, особенно обращая внимание на то, почему владелец источника хочет предоставить данные. Например, известно, что публикации о табачных изделиях и лекарствах очень часто оплачены табачными и фармацевтическими компаниями или их конкурентами. То же самое можно сказать о новостных компаниях, особенно тех, которые зависят от рекламных доходов или не раскрывают свою бизнес-модель.

Смещения выборки можно избежать, изучив реальную долю различных свойств данных, наблюдаемых в производственной среде, а затем включив в обучающий набор примеры в такой же пропорции.

Смещение из-за предубеждения или **стереотипов** можно контролировать. Разрабатывая модель для различения фотографий с изображением мужчин и женщин, аналитик мог бы уменьшить долю женщин на улице или увеличить долю мужчин в доме. Иными словами, чтобы уменьшить смещение из-за предубеждения или стереотипов, нужно предъявить алгоритму обучения более беспристрастное распределение примеров.

Систематическую ошибку можно снизить, расширив ассортимент измерительных устройств или наняв людей, умеющих сравнивать результаты измерений или наблюдений.

Необъективности экспериментатора можно избежать, поручив нескольким людям проверить задаваемые вопросы. Спросите себя: «Испытываю ли я неловкость или скованность, отвечая на этот вопрос?» Кроме того, несмотря на дополнительные трудности для анализа, отдавайте предпочтение открытым вопросам, а не вопросам, на которые нужно отвечать только да или нет или выбирать ответы из готового списка. Если вы все же решите ограничить выбор ответов, то хотя бы предусмотрите вариант «Прочее» и место для формулирования респондентом своего ответа.

Смещения пометки можно избежать, попросив нескольких разметчиков идентифицировать один и тот же пример. Спросите их, почему они решили назначить именно такую метку примерам, помеченным по-разному. Заметив, что некоторые разметчики ссылаются на определенные ключевые фразы, а не пытаются перефразировать весь документ, вы сможете выявить тех, кто лишь мельком просматривает, а не читает документы.

Можно также сравнить частоту документов, пропущенных разными разметчиками. Заметив, что какой-то разметчик пропускает документы чаще, чем в среднем, поинтересуйтесь, столкнулся ли он с техническими затруднениями или просто некоторые темы ему неинтересны.

Полностью избежать смещения данных невозможно. Панацеи не существует. Но есть общее правило – включайте в процесс человека, особенно если от модели зависят человеческие жизни.

Помните, что аналитики данных склонны предполагать, что моделям машинного обучения внутренне присуща честность, потому что они принимают решения, опираясь на факты и математику, в отличие от зачастую сумбурных и иррациональных человеческих суждений. Увы, не всегда дело обстоит так; модель, обученная на смещенных данных, неизбежно будет выдавать смещенные результаты.

Гарантировать справедливость выхода – обязанность людей, обучающих модель. Но что такое справедливость, спросите вы. И снова вынужден от-

ветить, что нет такого способа, который гарантированно определял бы несправедливость. Понятие справедливости модели всегда зависит от задачи, и только человек может его определить. В разделе 7.6 главы 7 мы рассмотрим несколько определений **справедливости** в машинном обучении.

Вмешательство человека на всех этапах сбора и подготовки данных – лучший способ минимизировать возможный ущерб от машинного обучения.

3.2.5. Низкая предсказательная способность

О проблеме **низкой предсказательной способности** часто задумываются только после того, как на обучение модели бесплодно потрачено много сил. Быть может, модель никак не удастся заставить работать, потому что она недостаточно выразительна? Быть может, в данных недостаточно информации для обучения? Мы не знаем.

Допустим, что наша цель – предсказать, понравится ли слушателю новая песня на музыкальном потоковом сервисе. Данные – имя исполнителя, название песни, текст и сведения о том, находится ли песня в плейлисте слушателя. Модель, обученная на этих данных, будет далека от совершенства.

Исполнители, отсутствующие в плейлисте слушателя, вряд ли получают от модели высокую оценку. На музыкальные предпочтения сильно влияют такие факторы, как аранжировка песни, выбор музыкальных инструментов, звуковые эффекты, тембр голоса и тонкие изменения тональности, ритма и темпа. Эти свойства произведения не найти ни в тексте, ни в названии, ни в имени исполнителя; их необходимо извлечь из самого музыкального файла.

С другой стороны, выделить эти релевантные признаки из аудиофайла нелегко. Даже при использовании современных нейронных сетей рекомендация песен на основе их звучания считается трудной задачей для искусственного интеллекта. Обычно рекомендации составляются путем сравнения плейлистов разных слушателей и нахождения таких, которые содержат похожие композиции.

Рассмотрим другой пример низкой предсказательной способности. Допустим, мы хотим обучить модель, которая будет предсказывать, куда направить телескоп, чтобы наблюдать какое-то интересное явление. Данные – фотографии различных участков небосвода, где в прошлом наблюдалось что-то необычное. Крайне маловероятно, что только на основе этих снимков мы сможем обучить модель верно предсказывать события. Но если добавить к ним измерения, полученные от различных датчиков, например измерения радиосигналов из разных зон или вспышки интенсивности космических лучей, то шансы на верные предсказания повысятся.

Ваша работа может оказаться особенно трудной, когда вы обрабатываете набор данных впервые. Если, несмотря на усложнение модели, никак не удастся получить приемлемые результаты, то пора задуматься о проблеме низкой предсказательной способности. Придумайте как можно больше дополнительных признаков (проявите изобретательность!). Рассмотрите косвенные источники данных, которые помогли бы обогатить векторы признаков.

3.2.6. Устаревшие примеры

В течение некоторого времени после развертывания модель обычно работает хорошо. Длительность этого периода зависит от моделируемого явления.

Обычно (мы будем обсуждать это в разделе 9.4) в производственной среде развертывается какая-то процедура контроля качества модели. Если обнаружено аномальное поведение, то для корректировки модели добавляются новые обучающие данные, после чего модель заново обучается и развертывается.

Часто ошибка объясняется конечностью обучающего набора. В таких случаях добавление обучающих примеров способно выправить ситуацию. Но на практике нередко бывает, что модель начинает допускать ошибки из-за **смены концепта**. Так называется фундаментальное изменение в статистической связи между признаками и меткой.

Допустим, что модель предсказывает, понравится ли пользователю какой-то контент на сайте. Со временем предпочтения пользователя могут измениться, например потому, что он повзрослел, или потому, что открыл для себя что-то новое (я, например, три года назад не слушал джаз, а теперь слушаю!). Примеры, включенные в обучающие данные в прошлом, уже не отражают его предпочтения и, вместо того чтобы улучшать качество модели, только ухудшают его. Это и есть смена концепта. Посмотрите, не наблюдается ли в модели нисходящий тренд качества на новых данных.

Исправьте модель, удалив устаревшие примеры из обучающих данных. Отсортируйте обучающие примеры по убыванию новизны. Определите дополнительный гиперпараметр – какой процент самых новых примеров использовать для переобучения модели – и настройте его путем **поиска на сетке** или какого-нибудь другого метода настройки гиперпараметров.

Смена концепта – пример более широкой проблемы, которая называется **сдвигом распределения**. Мы будем рассматривать настройку гиперпараметров и другие типы смены распределения в разделах 5.6 и 6.3.

3.2.7. Выбросы

Выбросами называются примеры, не похожие на большинство примеров в наборе данных. Что считать «непохожим», решает аналитик данных. Обычно непохожесть измеряется в терминах какой-то метрики, например **евклидова расстояния**.

Но на практике то, что кажется выбросом в исходном векторе признаков, может оказаться типичным примером в векторном пространстве признаков, преобразованном с помощью **ядерной функции**. Преобразование пространства признаков часто выполняется явно моделью на основе ядра, например **методом опорных векторов** (support vector machine – SVM), или неявно глубокой нейронной сетью.

Поверхностные алгоритмы, например линейная или логистическая регрессия, а также некоторые ансамблевые методы, например AdaBoost, особенно чувствительны к выбросам. SVM в одной из формулировок менее чувстви-

телен к выбросам: специальный штрафующий гиперпараметр регулирует влияние неправильно классифицированных примеров (которые часто оказываются выбросами) на **решающей границе**. Если величина штрафа велика, то алгоритм SVM полностью исключает выбросы из рассмотрения при построении решающей границы (воображаемой гиперплоскости, разделяющей положительные и отрицательные примеры). Если она слишком мала, то некоторые регулярные примеры могут оказаться не по ту сторону решающей границы, по которую нужно. Оптимальное значение этого гиперпараметра следует искать, применяя методы настройки гиперпараметров.

Достаточно сложная нейронная сеть может обучиться вести себя по-разному для каждого выброса в наборе данных, но при этом будет хорошо работать для регулярных примеров. Это нежелательно, потому что модель становится избыточно сложной. А чем модель сложнее, тем дольше она обучается и медленнее предсказывает, и, кроме того, хуже обобщается после развертывания.

Следует ли исключать выбросы из обучающих данных или лучше использовать алгоритмы и модели машинного обучения, робастные к выбросам, – предмет споров. Удаление примеров из набора данных считается неправильным с научной и методологической точек зрения, особенно если набор данных мал. С другой стороны, в контексте больших данных выбросы, как правило, не оказывают существенного влияния на модель. С практической точки зрения, если исключение некоторых обучающих примеров приводит к повышению качества модели на зарезервированных данных, то исключение может быть оправдано. Какие примеры считать кандидатами на исключение, решается на основе той или иной меры сходства. Современный подход к нахождению такой меры заключается в том, чтобы построить **автокодировщик** и использовать ошибку реконструкции¹ как меру сходства или несходства: чем больше ошибка реконструкции для данного примера, тем сильнее он непохож на остальной набор данных.

3.2.8. Просачивание данных

Просачивание данных, или **просачивание целей**, – проблема, затрагивающая несколько этапов жизненного цикла машинного обучения, от сбора данных до оценивания модели. В этом разделе я опишу только, как эта проблема проявляется на этапах сбора и подготовки данных, а в последующих главах – другие ее формы.

Просачивание данных при обучении с учителем – это непреднамеренное включение такой информации о целевом показателе, которая не должна быть доступна. Процесс такого «загрязнения» показан на рис. 3.8. Обучение на загрязненных данных приводит к чрезмерно оптимистическим ожиданиям относительно качества модели.

¹ Модель автокодировщика обучается реконструировать свои входы по вектору **погружения**. Гиперпараметры автокодировщика настраиваются так, чтобы минимизировать ошибку реконструкции на зарезервированных данных.

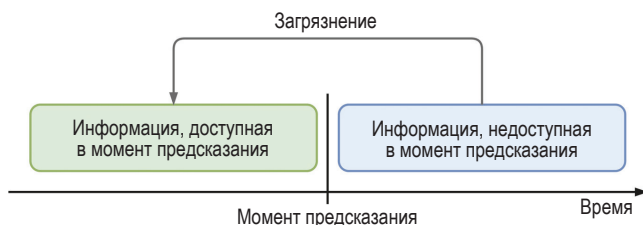


Рис. 3.8 ❖ Суть просачивания данных

3.3. Что считать хорошими данными

Мы уже обсуждали, какие вопросы о данных нужно задать, прежде чем приступать к их сбору, а также типичные проблемы, с которыми может столкнуться аналитик данных. Но что считать хорошими данными для проекта машинного обучения? Ниже мы рассмотрим несколько свойств хороших данных.

3.3.1. Хорошие данные информативны

Хорошие данные содержат достаточно информации, которую можно использовать для моделирования. Например, если мы хотим обучить модель, которая будет предсказывать, купит ли потребитель конкретный продукт, то понадобятся как свойства продукта, так и свойства тех продуктов, которые потребители покупали в прошлом. Если мы знаем только свойства продукта и имя и местоположение потребителя, то предсказания будут одинаковы для всех потребителей из одного и того же места.

При наличии достаточного количества обучающих примеров модель потенциально может вывести пол и этническую принадлежность из имени и давать разные предсказания для мужчин, женщин, различных местоположений и этносов, но не для каждого потребителя индивидуально.

3.3.2. Хорошие данные обладают хорошим покрытием

Хорошие данные достаточно полно покрывают наши намерения относительно применения модели. Например, если мы собираемся использовать модель для классификации веб-страниц по теме и имеется тысяча интересующих нас тем, то данные должны содержать примеры документов по каждой из тысячи тем в количестве, достаточном, чтобы алгоритм мог обучиться различию между темами.

Рассмотрим другую ситуацию. Предположим, что по некоторой теме имеется всего один или два документа. Допустим, что в тексте каждого документа присутствует уникальный идентификатор. В таком случае обучающий

алгоритм не сможет понять, на что смотреть в документе, чтобы определить, к какой теме он относится. Быть может, на идентификаторы? Вроде бы они хорошо определяют различия. Но если алгоритм решит использовать идентификаторы, чтобы отделить эти два примера от остального набора данных, то обученная модель будет неспособна к обобщению: она никогда не увидит эти идентификаторы снова.

3.3.3. Хорошие данные отражают реальные входы

Хорошие данные отражают реальные входы, которые модель будет видеть в производственной среде. Например, если мы строим систему для распознавания автомобилей на дороге, а все фотографии были сделаны в рабочее время, то вряд ли у нас наберется много примеров ночной дороги. Но развернутой модели фотографии будут поступать в любое время суток, и для ночных снимков она часто будет допускать ошибки. Кроме того, не забывайте о проблеме кошки, собаки и енота: если модель ничего не знает о енотах, то она будет классифицировать их фотографии как кошку или собаку.

3.3.4. Хорошие данные несмещенные

Хорошие данные должны быть по возможности несмещенными. На первый взгляд, это свойство похоже на предыдущее. Но смещение может присутствовать как в данных, используемых для обучения, так и в данных, к которым модель применяется в процессе эксплуатации.

В разделе 3.2 мы обсудили несколько источников смещения данных и как с ними бороться. Источником смещения может быть также пользовательский интерфейс. Например, пусть мы хотим предсказать популярность новостной статьи и в качестве признаков используем число переходов по ссылке. Если какая-то новость отображается в верхней части страницы, то количество переходов на нее часто оказывается больше, чем на новость внизу страницы, пусть даже последняя более интересна.

3.3.5. Хорошие данные не являются результатом петли обратной связи

Хорошие данные не должны быть результатом самой модели. Это возвращает нас к проблеме петли обратной связи, обсуждавшейся выше. Например, нельзя обучить модель, предсказывающую пол человека по его имени, а затем использовать это предсказание для пометки нового обучающего примера.

Еще пример – если модель используется, чтобы решить, какие сообщения электронной почты важны пользователю, и подсветить важные сообщения, то не следует воспринимать щелчок по такому сообщению как сигнал о том, что оно важно. Возможно, пользователь щелкнул по нему как раз потому, что модель его подсветила.

3.3.6. У хороших данных согласованные метки

Хорошие данные должны иметь согласованные метки. Несогласованность пометки может проистекать из нескольких источников:

- разные люди помечают примеры, пользуясь разными критериями. Даже если они полагают, что используют один и тот же критерий, может случиться, что разные люди интерпретируют этот критерий по-разному¹;
- определения некоторых классов со временем изменяются. В результате возникает ситуация, когда очень похожие векторы признаков получают разные метки;
- ошибочная интерпретация мотивов пользователя. Предположим, к примеру, что пользователь проигнорировал рекомендованную новость. Как следствие эта новость получает отрицательную метку. Но ведь могло случиться, что пользователь проигнорировал рекомендацию просто потому, что уже знает эту историю, а не потому, что не интересуется данной тематикой.

3.3.7. Хорошие данные достаточно велики

Хорошие данные достаточно велики, чтобы обеспечить обобщаемость. Иногда ничто не помогает улучшить верность модели. Не важно, сколько данных бросать в топку алгоритма: информация, содержащаяся в данных, не обладает нужной для задачи предсказательной способностью. Но чаще можно получить очень верную модель, если передать ей не тысячи, а миллионы или сотни миллионов примеров. Сколько понадобится данных, невозможно сказать заранее: нужно начать работать над задачей и наблюдать за прогрессом.

3.3.8. Сводный перечень свойств хороших данных

Чтобы удобнее было ссылаться, еще раз перечислим свойства хороших данных:

- содержат достаточно информации, которую можно использовать для моделирования;
- довольно полно покрывают намерения относительно применения модели;
- отражают реальные входные данные, которые модель будет видеть на этапе эксплуатации;
- максимально несмещенные;
- не являются результатом самой модели;
- метки согласованы;
- данные достаточно велики для обобщаемости.

¹ Вспомните пример механического турка из раздела 3.1. Для повышения надежности меток, присвоенных разными людьми, можно использовать большинство голосов или усреднять предложения нескольких разметчиков.

3.4. ОБРАБОТКА ДАННЫХ О ВЗАИМОДЕЙСТВИИ

Данные о взаимодействии – это данные, которые собираются в процессе взаимодействия пользователей с системой, поддерживаемой моделью. Считайте, что вам повезло, если вы можете собрать хорошие данные на основе взаимодействия пользователей с системой.

Хорошие данные о взаимодействии содержат информацию трех видов:

- контекст взаимодействия;
- действие пользователя в этом контексте;
- результат взаимодействия.

Например, предположим, что мы создаем поисковую систему, и модель ранжирует результаты поиска для каждого пользователя индивидуально. Модель ранжирования принимает на входе список ссылок, возвращенный поисковой системой на основе ключевых слов, заданных пользователем, и выводит другой список, в котором порядок ссылок изменен. Обычно модель ранжирования что-то «знает» о пользователе и его предпочтениях и может переупорядочить результаты поиска для каждого пользователя индивидуально в соответствии с его предпочтениями. Контекстом здесь является поисковый запрос и сотни документов, предъявленных пользователю в определенном порядке. Действие – это переход по ссылке на конкретный документ. Результат – сколько времени пользователь потратил на чтение документа и нажал ли после этого кнопку «Назад». Еще одно действие – переход по ссылке «Следующая страница».

Интуитивно кажется, что ранжирование хорошее, если пользователь перешел по ссылке и достаточно долго читал страницу. Ранжирование не очень хорошее, если пользователь перешел по ссылке и быстро вернулся обратно. Ранжирование плохое, если пользователь перешел по ссылке «Следующая страница». Эти данные можно использовать, чтобы улучшить алгоритм ранжирования и сделать его более персонализированным.

3.5. ПРИЧИНЫ ПРОСАЧИВАНИЯ ДАННЫХ

Обсудим три самые частые причины **просачивания данных** во время их сбора и подготовки: 1) цель является функцией от признака; 2) признак скрывает цель; 3) признак берется из будущего.

Страна	Население	Регион	...	ВВП на душу населения	ВВП
Франция	67M	Европа	...	38 800	2,6Т
Германия	83M	Европа	...	44 578	3,7Т
...
Китай	1386M	Азия	...	8802	12,2Т

Рис. 3.9 ❖ Пример целевого показателя (ВВП), являющегося простой функцией от двух признаков: население и ВВП на душу населения

3.5.1. Цель является функцией от признака

Валовой внутренний продукт (ВВП) определяется как рыночная стоимость всех конечных товаров и услуг, произведенных в стране за определенный период. Допустим, что мы хотим предсказать ВВП страны на основе различных атрибутов: площадь, численность населения, географический регион и т. д. Пример таких данных приведен на рис. 3.9. Если анализ атрибутов и их связи с ВВП проведен небрежно, то может произойти просачивание данных: на рис. 3.9 произведение столбцов «Население» и «ВВП на душу населения» равно ВВП. Обученная модель будет идеально предсказывать ВВП, глядя только на эти два столбца. Мы допустили, что ВВП является одним из признаков, пусть и немного модифицированным (поделенным на численность населения), и тем самым создали загрязнение, ставшее причиной просачивания данных.

Еще более простой пример – когда копия целевого показателя уже присутствует среди признаков, только в другом формате. Допустим, что мы обучаем модель, которая должна предсказывать годовую зарплату по атрибутам работника. В роли обучающих данных выступает таблица, содержащая среди прочего месячную и годовую зарплаты. Если вы позабудете исключить месячную зарплату из списка признаков, то одного этого атрибута будет достаточно для точного предсказания годовой зарплаты, и вы поверите, что модель идеальная. Но стоит развернуть ее в производственной среде, как она перестанет получать информацию о месячной зарплате работника, а иначе зачем бы одна была нужна?

3.5.2. Признак скрывает цель

Иногда целевой показатель не является функцией от одного или нескольких признаков, а «скрыт» в одном из признаков. Рассмотрим набор данных на рис. 3.10.

ИД клиента	Группа	Годовые расходы	Просмотров страниц в год	...	Пол
1	M18-25	1350	11 987	...	М
2	Ж25-35	2365	44 578	...	Ж
...
18879	Ж65+	3653	6775		Ж

Рис. 3.10 ❖ Пример цели, скрытой в одном из признаков

Здесь мы используем данные о клиенте для предсказания его пола. Рассмотрим столбец «Группа». Внимательно изучив данные в этом столбце, мы обнаружим, что он содержит демографический показатель, с которым каждый из существующих клиентов был соотнесен в прошлом. Если данные о поле и возрасте клиента фактические (а не выданные другой моделью, развернутой в производственной среде), то столбец «Группа» дает пример

просачивания данных, когда значение, которое мы хотим предсказывать, «скрыто» в значении признака.

С другой стороны, если в столбце «Группа» находятся значения, предсказанные другой, возможно, менее верной моделью, то этот атрибут можно использовать для построения более сильной модели. Эта техника называется **штабелированием моделей** (model stacking), мы рассмотрим ее в разделе 6.2.

3.5.3. Признак из будущего

Признак из будущего – это форма просачивания данных, которую трудно обнаружить, не имея ясного представления о бизнес-цели. Допустим, клиент просит обучить модель, которая будет предсказывать, вернет ли заемщик долг, на основании таких атрибутов, как возраст, пол, образование, зарплата, семейное положение и т. д. Пример данных приведен на рис. 3.11.

ИД заемщика	Группа	Образование	...	Напоминаний о просроченном платеже	Погасит заем
1	M35-50	Школа	...	0	Д
2	Ж25-35	Магистр	...	1	Н
...
65723	M25-35	Магистр		3	Н

Рис. 3.11 ❖ Признак, недоступный в момент предсказания:
«Напоминаний о просроченном платеже»

Не предприняв усилий для осмысления контекста, в котором будет использоваться модель, вы, возможно, решите использовать все доступные атрибуты для предсказания значения в столбце «Погасит заем», включая и данные из столбца «Напоминаний о просроченном платеже». Модель будет казаться верной на этапе тестирования, и вы передадите ее клиенту, который затем сообщит, что модель не работает в производственной среде.

Разобравшись, вы обнаруживаете, что в производственной среде атрибут «Напоминаний о просроченном платеже» всегда равен нулю. Это и понятно, потому что клиент пользуется вашей моделью, до того как заемщик взял кредит, так что никаких напоминаний еще не было! Но модель-то, скорее всего, обучилась выдавать предсказание «Нет», когда количество напоминаний больше или равно 1, и на остальные признаки обращает меньше внимания.

Вот еще один пример. Допустим, у вас имеется сайт и вы хотите ранжировать показываемые пользователю новости, так чтобы максимизировать число переходов по ссылкам. Если в составе обучающих данных присутствуют позиционные признаки для каждой новости, показанной прежде (например, позиция x –у заголовка и блока краткого содержания на странице), то во время показа эта информация будет недоступна, потому что до момента ранжирования позиции новостей на странице неизвестны.

Таким образом, понимать, в каком контексте будет использоваться модель, очень важно для предотвращения просачивания данных.

3.6. РАЗБИЕНИЕ ДАННЫХ

В разделе 1.3.3 первой главы мы говорили, что на практике при выполнении машинного обучения имеется три непересекающихся набора примеров: обучающий, контрольный и тестовый.

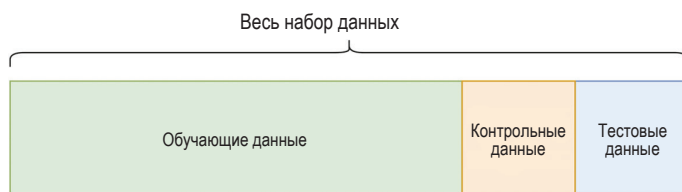


Рис. 3.12 ❖ Весь набор данных разбит на три части: обучающий, контрольный и тестовый

Обучающий набор используется алгоритмом машинного обучения для обучения модели.

Контрольный набор нужен, чтобы найти оптимальные значения гиперпараметров конвейера машинного обучения. Аналитик поочередно пробует разные комбинации гиперпараметров, обучает модель с применением каждой комбинации и смотрит, какое качество она показывает на контрольном наборе. Гиперпараметры, при которых качество максимально, используются для обучения производственной модели. Методы настройки гиперпараметров мы рассмотрим более подробно в разделе 5.6.

Тестовый набор используется для отчета: выбрав наилучшую модель, мы тестируем ее качество на тестовом наборе и протоколируем результаты.

Контрольный и тестовый наборы часто называют **зарезервированными наборами**: они содержат примеры, которые не должны предъявляться алгоритму обучения.

Хорошее разбиение набора данных на три непересекающиеся части должно удовлетворять нескольким условиям.

Условие 1. Разбиение применялось к первичным данным.

Получив первичные примеры сразу, прежде чем делать что-то еще, произведите разбиение. Ниже мы увидим, как это помогает избежать просачивания данных.

Условие 2. Перед разбиением данные были рандомизированы.

Случайным образом перетасуйте примеры, затем произведите разбиение.

Условие 3. Контрольный и тестовый наборы распределены одинаково.

Выбирая наилучшие значения гиперпараметров с помощью контрольного набора, мы хотим, чтобы полученная модель хорошо работала в производственной среде. Примеры из тестового набора – типичные представители

производственных данных. Поэтому и нужно, чтобы распределение контрольного и тестового наборов было одинаково.

Условие 4. При разбиении удалось избежать просачивания.

Просачивание данных может случиться даже во время разбиения данных. Ниже мы увидим, какие именно формы просачивания возможны на этой стадии.

Не существует идеального соотношения размера трех наборов. В старой литературе (до эры больших данных) можно было встретить рекомендацию производить разбиение в пропорции 70:15:15 или 80:10:10 (доли обучающего, контрольного и тестового наборов соответственно).

Но теперь, в эпоху интернета и дешевого труда (например, механический турок или краудсорсинг), организации, ученые и даже энтузиасты у себя дома могут получить доступ к миллионам обучающих примеров. Поэтому расточительно использовать для обучения только 70 или 80 % доступных данных.

Контрольные и тестовые данные используются лишь для вычисления статистики, отражающей качество модели. Их размер должен быть достаточен для получения надежной статистики – и только. Сколько это – вопрос дискуссионный. Навскидку можно предположить, что дюжина примеров на класс – желательный минимум. Если вы можете позволить себе сотню примеров на класс в каждом из двух зарезервированных наборов, то запас прочности хороший, и статистика, вычисленная по таким наборам, будет надежной.

Пропорции могут также зависеть от выбранного алгоритма обучения или модели. Качество моделей глубокого обучения обычно заметно улучшается, если подать им больше обучающих данных. Для поверхностных алгоритмов и моделей зависимость не такая прямая.

Кроме того, пропорции могут зависеть от размера набора данных. Если набор невелик и содержит меньше тысячи примеров, то лучше использовать 90 % данных для обучения. В таком случае можно не заводить отдельного контрольного набора, а имитировать его с помощью техники **перекрестной проверки**. Мы рассмотрим этот вопрос в разделе 5.6.5.

Отметим, что при разбиении на три части **данных временных рядов** нужно позаботиться о сохранении порядка наблюдений в каждом примере в процессе перемешивания. Иначе для большинства задач предсказания данные окажутся некорректными и обучить модель не получится. О временных рядах мы будем говорить в разделе 4.2.6.

3.6.1. Просачивание во время разбиения

Как вы уже знаете, просачивание данных может произойти на любой стадии, от сбора данных до оценивания модели. И стадия разбиения данных не исключение.

Групповое просачивание может случиться во время разбиения. Допустим, имеются магнитно-резонансные изображения головного мозга нескольких пациентов. Каждое изображение помечено определенным заболеванием, а некоторые пациенты могут быть представлены несколькими изображениями, сделанными в разное время. Если применить описанный

выше метод разбиения (перемешать, затем разбить), то изображения одного пациента могут оказаться и в обучающем, и в зарезервированном наборах.

Модель могла бы обучиться на особенностях пациента, а не болезни. Модель запомнила бы, что в мозге пациента А имеется определенный рисунок извилин, и если в обучающих данных этот пациент помечен каким-то заболеванием, то модель успешно предскажет это заболевание в контрольных данных, опознав пациента А по одному лишь рисунку извилин.

Решением проблемы является **групповое разбиение**. Это значит, что все примеры, относящиеся к одному пациенту, нужно собрать в одном наборе: обучающем или зарезервированном. И это еще раз подтверждает, как важно для аналитика знать о данных максимально много.

3.7. ОБРАБОТКА ОТСУТСТВИЯ АТТРИБУТОВ

Иногда данные поступают аналитику в аккуратной форме, например в виде электронной таблицы Excel¹, но некоторые атрибуты отсутствуют. Это часто бывает, когда набор данных составлялся вручную и человек забыл включить или не измерил какие-то значения.

Перечислим типичные подходы к обработке отсутствия значений атрибута:

- удалить примеры с отсутствующими атрибутами из набора данных (это допустимо, если набор данных достаточно велик, чтобы пожертвовать частью данных);
- использовать алгоритм обучения, который умеет обрабатывать отсутствие значений атрибутов (например, алгоритм обучения решающих деревьев);
- применить методы **подстановки данных**.

3.7.1. Методы подстановки данных

Чтобы подставить значение отсутствующего числового атрибута, можно, например, взять среднее его значений в других примерах. Математически это выглядит следующим образом. Пусть атрибут j отсутствует в некоторых примерах исходного набора данных, а $\mathcal{S}^{(j)}$ – множество размера $N^{(j)}$, содержащее только те примеры из исходного набора, в которых атрибут j присутствует. Тогда вместо отсутствующего атрибута j подставляется следующее значение $\hat{x}^{(j)}$:

$$\hat{x}^{(j)} \leftarrow \frac{1}{N^{(j)}} \sum_{i \in \mathcal{S}^{(j)}} x_i^{(j)},$$

¹ Из того, что первичный набор данных хранится в таблице Excel, еще не следует, что данные аккуратные. Одно из условий аккуратности – каждая строка должна представлять один пример.

где $N^{(j)} < N$, а суммирование производится только по тем примерам, в которых присутствует значение атрибута j . На рис. 3.13 показано, как работает этот метод, когда в двух примерах (в строках 1 и 3) отсутствует значение атрибута «Рост». В пустые ячейки подставляется среднее значение 177.

Строка	Возраст	Вес	Зарплата	Рост
1	18	70		35 000
2	43	65	175	26 900
3	34	87		76 500
4	21	66	187	94 800
5	65	60	169	19 000

$Height \leftarrow \frac{1}{3}(175 + 187 + 169) = 177$

Рис. 3.13 ❖ Замена отсутствующего значения средним значением атрибута по набору данных

Еще один способ – подставить вместо отсутствующего значение, выходящее за предел обычного для данного атрибута диапазона. Например, если атрибут принадлежит отрезку $[0, 1]$, то вместо отсутствующего значения можно подставить 2 или -1 ; если же атрибут категориальный, например день недели, то вместо отсутствующего значения можно подставить «Неизвестно» (Unknown). Тогда алгоритм сам обучится, что делать, когда значение атрибута отличается от нормально допустимых. Для числовых атрибутов существует еще один способ замены – подставить среднее значение в диапазоне. Например, если диапазон значений атрибута – $[-1, 1]$, то вместо отсутствующего значения можно подставлять 0. Идея в том, что среднее значение не должно существенно влиять на предсказание.

Более продвинутый метод – использовать отсутствующее значение как целевую переменную в задаче регрессии (предполагается, что все атрибуты числовые). Остальные атрибуты $[x_i^{(1)}, x_i^{(2)}, \dots, x_i^{(j-1)}, x_i^{(j+1)}, \dots, x_i^{(D)}]$ можно использовать для формирования вектора признаков $\hat{\mathbf{x}}_i$ и положить $\hat{y}_i \leftarrow x_i^{(j)}$, где j – атрибут с отсутствующим значением. После этого мы строим модель регрессии, предсказывающую \hat{y} по $\hat{\mathbf{x}}$. Конечно, при построении обучающих примеров $(\hat{\mathbf{x}}, \hat{y})$ используются только примеры из исходного обучающего набора, в которых значение атрибута j присутствует.

Наконец, если имеется достаточно большой набор данных и всего несколько атрибутов с отсутствующими значениями, то можно добавить синтетический бинарный индикаторный атрибут для каждого исходного атрибута с отсутствующими значениями. Предположим, что примеры в наборе данных D -мерные, а у атрибута в позиции $j = 12$ некоторые значения отсутствуют. Тогда для каждого примера \mathbf{x} мы добавляем в позицию $j = D + 1$ атрибут, равный 1, если значение атрибута в позиции 12 отсутствует в \mathbf{x} , и 0 в противном случае. Вместо самого отсутствующего значения можно подставить 0 или любое другое число, какое вам нравится.

В момент предсказания, если пример неполон, то следует применить тот же метод подстановки отсутствующих значений, что при пополнении обучающих данных.

Перед началом работы над задачей обучения нельзя сказать, какой метод подстановки будет работать лучше. Попробуйте несколько вариантов, постройте несколько моделей и выберите из них наилучшую (для сравнения моделей используйте контрольный набор).

3.7.2. Просачивание во время подстановки

Если для подстановки применяется метод, вычисляющий какую-то статистику одного атрибута (например, среднее) или нескольких атрибутов (путем решения задачи регрессии), то просачивание будет иметь место, если для вычисления этой статистики используется весь набор данных. Работая со всеми доступными примерами, мы загрязняем обучающие данные информацией, полученной из контрольных и тестовых примеров.

Этот вид просачивания не так важен, как другие, рассмотренные выше. Но все равно о нем нужно знать и избегать такого развития событий – для этого сначала разбейте данные на части, а затем вычисляйте статистику для подстановки только на обучающем наборе.

3.8. ПРИРАЩЕНИЕ ДАННЫХ

Для некоторых типов данных очень легко получить новые помеченные примеры без дополнительной разметки. Эта стратегия называется **приращением данных** и особенно эффективна применительно к изображениям. Она подразумевает применение простых операций, например кадрирования или отражения, к оригинальным изображениям с целью получения новых.

3.8.1. Приращение данных для изображений

На рис. 3.14 приведены примеры операций, которые легко применить к заданному изображению для получения новых: отражение, поворот, кадрирование, замена цвета, добавление шума, изменение перспективы, изменение контрастности и потери части информации.

Разумеется, отражение относительно оси следует производить так, чтобы сохранился смысл изображения. Футбольный мяч можно отразить относительно обеих осей¹, но изображение автомобиля или пешехода – только относительно вертикальной.

Поворачивать следует на небольшой угол, чтобы имитировать неправильную установку горизонта. Поворачивать изображение можно в обоих направлениях.

Кадрирование можно применять несколько раз к одному и тому же изображению случайным образом, сохраняя значительные части представляющих интерес объектов в кадре.

¹ Если только контекст, например трава на газоне, не делает отражение относительно горизонтальной оси бессмысленным.



Рис. 3.14 ❖ Примеры методов приращения данных.
Фотографировал Альфонсо Эскаланте

В случае замены цвета красная, зеленая и синяя компоненты (RGB) немного изменяются, чтобы имитировать различные условия освещения. Изменение контрастности (как уменьшение, так и увеличение) и **гауссов шум** различной интенсивности также можно применять несколько раз к одному и тому же изображению.

Путем случайного удаления частей изображения можно имитировать ситуации, когда объект можно распознать, но он не целиком виден из-за препятствия.

Еще одна популярная техника приращения данных, которая кажется противоречащей интуиции, но хорошо работает на практике, – **смешивание**. Как следует из названия, смысл в том, чтобы обучать модель на смеси изображений из обучающего набора. Точнее, вместо обучения модели на первичных изображениях мы берем два изображения (которые могут принадлежать одному классу или разным) и используем для обучения их линейную комбинацию:

$$\text{mixup_image} = t \times \text{image}_1 + (1 - t) \times \text{image}_2,$$

где t – вещественное число от 0 до 1. Целевым показателем для такого смешанного изображения является такая же линейная комбинация исходных целевых показателей:

$$\text{mixup_target} = t \times \text{target}_1 + (1 - t) \times \text{target}_2.$$

Эксперименты¹ на наборах данных **ImageNet-2012**, **CIFAR-10** и некоторых других показали, что смешивание улучшает обобщаемость нейросетевых моделей. Авторы этой техники также обнаружили, что она повышает робастность к **сопоставительным примерам** и устойчивость обучения **порождающих сопоставительных сетей** (generative adversarial network – GAN).

В дополнение к методам, показанным на рис. 3.14, если вы ожидаете, что входные изображения в производственной системе будут плотно сжаты, то можно имитировать такое сжатие, применяя некоторые часто используемые методы сжатия с потерей информации и соответствующие файловые форматы, например JPEG или GIF.

Приращению подвергаются только обучающие данные. Конечно, не имеет смысла генерировать все дополнительные примеры заранее и хранить их. На практике методы приращения данных применяются к исходным данным динамически во время обучения.

3.8.2. Приращение данных для текста

С приращением текстовых данных все обстоит не так просто. Необходимо использовать методы преобразования, сохраняющие контекстуальную и грамматическую структуры текстов на естественном языке.

Допустим, можно заменять случайные слова в предложении близкими **синонимами**. Например, вот несколько предложений, эквивалентных «Машина остановилась возле торгово-развлекательного комплекса»:

Автомобиль остановился возле торгово-развлекательного комплекса.

Машина остановилась возле торгового центра.

Легковушка остановилась возле гипермаркета.

Похожая техника – использовать **гиперонимы** вместо синонимов. Гиперонимом называется слово с более общим значением. Например, «млекопитающее» является гиперонимом для слов «кит» и «кот»; «транспортное средство» – для «автомобиль» и «автобус». В нашем примере можно было бы создать такие предложения:

Транспортное средство остановилось возле торгового центра.

Машина остановилась возле здания.

Если слова или документы в наборе данных представлены с помощью **погружений слов** или **документов**, то можно применить слабый гауссов шум к случайно выбранным признакам погружения, чтобы создать вариант того же слова или документа. Количество подлежащих модификации признаков

¹ Дополнительные сведения о технике смешивания см. в работе *Zhang Hongyi, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. arXiv preprint arXiv:1710.09412 (2017).*

и интенсивность шума можно настраивать как гиперпараметры путем оптимизации качества на контрольном наборе.

Можно поступить по-другому – для замены слова w в предложении найти k ближайших его соседей в пространстве погружения слов и сгенерировать k новых предложений, подставив вместо слова w соответствующего соседа. Для поиска ближайших соседей можно применить такие метрики, как **коэффициент Отиаи** (косинусный коэффициент) или **евклидово расстояние**. Выбор метрики и величину k можно настроить как гиперпараметры.

Современная альтернатива методу k ближайших соседей – использовать глубокую предобученную модель, например Bidirectional Encoder Representations from Transformers (**BERT**). Модели типа BERT обучаются предсказывать замаскированное слово по другим словам в предложении. С помощью BERT можно сгенерировать k наиболее вероятных предсказаний замаскированного слова, а затем использовать их как синонимы для приращения данных.

Аналогично в случае задачи классификации документов, когда имеется большой корпус непомеченных и малый корпус помеченных документов, можно поступить следующим образом. Сначала построим погружения для всех документов в большом корпусе, для чего можно воспользоваться методом **doc2vec** или каким-то другим. Затем для каждого помеченного документа d в наборе данных найдем k ближайших непомеченных документов в пространстве погружения и пометим их той же меткой, что и d . Значение k подбирается с помощью контрольных данных.

Еще одна полезная техника приращения данных – **обратный перевод**. Чтобы создать новый пример англоязычного текста (предложение или документ), сначала переведем его на другой язык l , воспользовавшись системой машинного перевода. Затем выполним обратный перевод с l на английский. Если получившийся текст отличается от оригинального, добавим его в набор данных, пометив той же меткой, что оригинальный.

Существуют также методы приращения данных для других типов данных, например аудио или видео: добавление шума, сдвиг аудио или видеоролика во времени, замедление или ускорение, изменение высоты звука в случае аудио или цветового баланса в случае видео и т. д. Детальное описание этих методов выходит за рамки книги. Просто имейте в виду, что приращение можно применить к любым мультимедийным данным, а не только к изображениям или тексту.

3.9. ОБРАБОТКА НЕСБАЛАНСИРОВАННЫХ ДАННЫХ

Несбалансированность классов может оказать существенное влияние на качество модели независимо от выбранного алгоритма обучения. Суть проблемы – крайне неравномерное распределение меток в обучающих данных.

Так бывает, например, когда классификатор должен отличать настоящие электронные транзакции от мошеннических: примеров настоящих транзакций гораздо больше. Типичный алгоритм машинного обучения старается

классифицировать большинство обучающих примеров правильно. Он вынужден так делать, потому что предназначен для минимизации **функции стоимости**, которая обычно назначает положительную потерю каждому неправильно классифицированному примеру. Если потеря при неправильной классификации примеров редко встречающихся классов такая же, как для часто встречающихся, то может случиться, что алгоритм обучения сочтет выгодным «сдаваться» на многих примерах редко встречающихся классов, чтобы делать меньше ошибок на часто встречающихся.

Формального определения **несбалансированных данных** не существует, но можно рассмотреть следующее эвристическое правило. Если классов два, то сбалансированность означает, что каждый класс представлен половиной набора данных. Небольшая несбалансированность обычно не составляет проблемы. Так, если 60 % принадлежат одному классу, а 40 % другому и мы используем какой-нибудь популярный алгоритм машинного обучения в стандартной формулировке, то, как правило, качество модели не должно сильно пострадать. Но если несбалансированность классов велика, например одному классу принадлежит 90 % примеров, а другому только 10 %, то стандартная формулировка, которая обычно назначает одинаковый вес ошибкам на примерах обоих классов, может оказаться неэффективной, так что потребуются ее модифицировать.

3.9.1. Выборка с избытком

Чтобы сгладить проблему несбалансированности классов, часто применяют технику **выборки с избытком** (oversampling). Если сделать несколько копий примеров редко встречающегося класса, как показано на рис. 3.15 слева, то их вес повысится. Можно также создавать синтетические примеры – для этого выберем значения признаков из нескольких примеров редко встречающегося класса и, комбинируя их, построим новый пример этого класса. Есть два популярных алгоритма выборки с избытком путем синтеза примеров редко встречающегося класса: Synthetic Minority Oversampling Technique (**SMOTE**) и Adaptive Synthetic Sampling Method (**ADASYN**).

Методы SMOTE и ADASYN во многих отношениях похожи. Для заданного примера \mathbf{x}_i редко встречающегося класса оба метода выбирают k ближайших соседей. Обозначим этот набор k примеров \mathcal{S}_k . Синтетический пример \mathbf{x}_{new} определяется как $\mathbf{x}_i + \lambda(\mathbf{x}_{z_i} - \mathbf{x}_i)$, где \mathbf{x}_{z_i} – пример редко встречающегося класса, случайно выбранный из \mathcal{S}_k . Гиперпараметр интерполяции λ может быть произвольным числом из отрезка $[0, 1]$. (На рис. 3.16 приведена иллюстрация для $\lambda = 0.5$.)

И SMOTE, и ADASYN случайно выбирают любой пример \mathbf{x}_i из набора данных. В ADASYN количество синтетических примеров, генерируемых для каждого \mathbf{x}_i , пропорционально числу примеров в \mathcal{S}_k , не принадлежащих редко встречающемуся классу. Поэтому больше синтетических примеров генерируются в той области, где примеры редко встречающегося класса действительно редки.

3.9.2. Выборка с недостатком

Противоположный подход – **выборка с недостатком** (undersampling) – заключается в том, чтобы исключить из обучающего набора некоторые примеры часто встречающегося класса (рис. 3.15, справа).

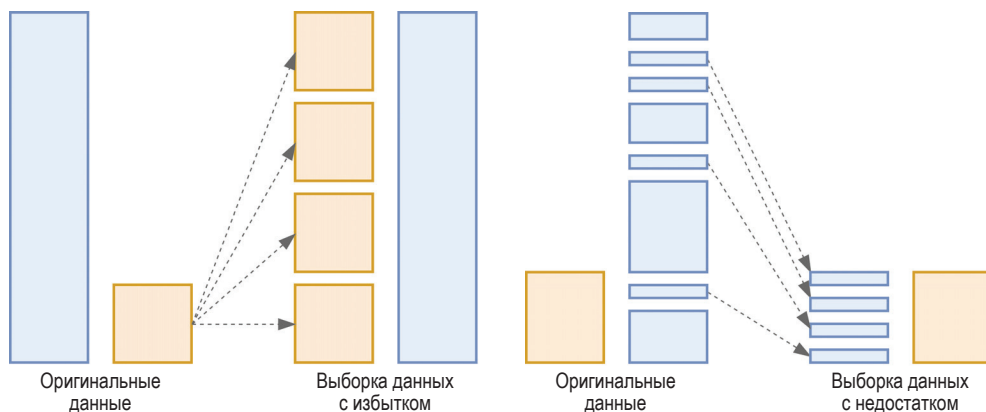


Рис. 3.15 ❖ Выборка с избытком (слева) и с недостатком (справа)

Выборку с недостатком можно производить случайно, т. е. случайным образом выбирать примеры, подлежащие удалению из часто встречающегося класса. Альтернативно можно исключать примеры, обладающие определенным свойством. Одно из таких свойств называется **связями Томека**. Связь Томека между примерами x_i и x_j , принадлежащими двум разным классам, существует, если в наборе данных нет примера x_k , расстояние от которого до x_i или x_j меньше, чем расстояние между ними самими. Расстояние можно определить, например, как коэффициент Отиаи или евклидово расстояние.

На рис. 3.17 показано, как удаление примеров из часто встречающегося класса на основе связей Томека помогает организовать отчетливый зазор между примерами обоих классов.

Кластерная выборка с недостатком устроена следующим образом. Решите, сколько примеров должно быть в часто встречающемся классе в результате выборки с недостатком. Обозначим это число k . Примените **алгоритм кластеризации на основе центроидов** к примерам часто встречающегося класса, задав в качестве количества кластеров k . Затем замените все примеры мажоритарных классов k центроидами. Одним из алгоритмов кластеризации на основе центроидов является алгоритм **k ближайших соседей**.

3.9.3. Гибридные стратегии

Гибридные стратегии (сочетание выборки с избытком и с недостатком) иногда позволяют улучшить результат. Одна из таких стратегий заключается в использовании ADASYN для выборки с избытком и связей Томека для выборки

с недостатком. Другая стратегия – комбинация кластерной выборки с недостатком и метода SMOTE.

3.10. СТРАТЕГИИ ВЫБОРКИ ДАННЫХ

Если имеется большой информационный актив, так называемые большие данные, то не всегда практически полезно и необходимо работать со всем набором. Иногда можно произвести небольшую выборку, содержащую достаточно информации для обучения.

Аналогично, когда производится выборка с недостатком из часто встречающегося класса для корректировки несбалансированности данных, уменьшенная выборка должна быть репрезентативна для всего частого класса. В этом разделе мы обсудим несколько стратегий выборки, их свойства, преимущества и недостатки.

Существует две основные стратегии: вероятностная и невероятностная выборка. В случае **вероятностной выборки** шанс быть выбранным есть у любого примера. Такие методы подразумевают рандомизацию.

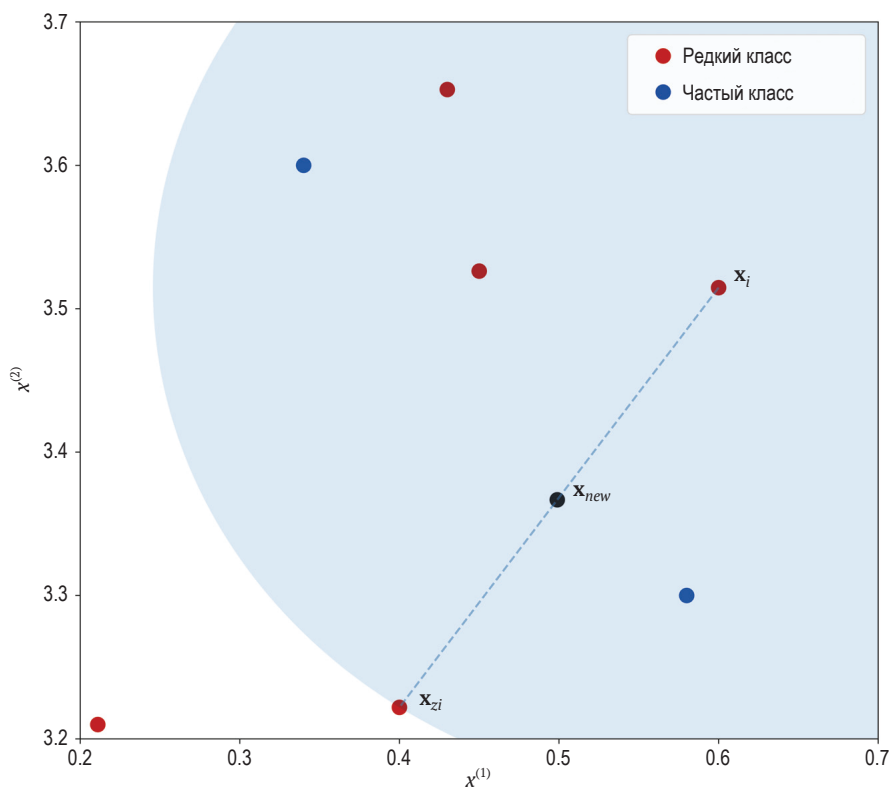


Рис. 3.16 ❖ Генерирование синтетических примеров с помощью комбинации SMOTE и ADASYN (построено модифицированным скриптом Гийома Лемэтра)

Для построения **невероятностной выборки** производится фиксированная детерминированная последовательность эвристических действий. Это означает, что у некоторых примеров нет шансов быть выбранными, сколько бы выборку мы ни производили.

Исторически невероятностные методы были удобнее для применения человеком вручную. Но сейчас это преимущество утратило силу. Аналитики данных пользуются компьютерами и программами, что существенно упрощает выборку даже из больших данных. Основной недостаток невероятностных методов – то, что они иногда строят нерепрезентативные выборки и могут систематически исключать важные примеры. Эти недостатки перевешивают их достоинства. Поэтому в данной книге я опишу только вероятностные методы выборки.

3.10.1. Простая случайная выборка

Простая случайная выборка – самый прямолинейный метод, который я буду подразумевать, говоря «произведена случайная выборка». Каждый пример выбирается из всего набора данных чисто случайно, у всех примеров равные шансы быть выбранными.

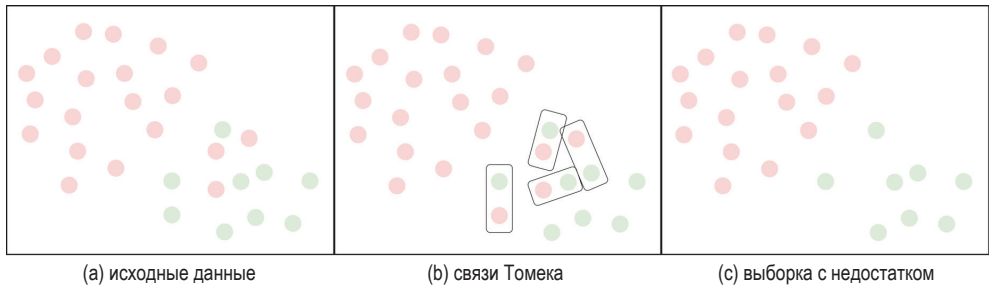


Рис. 3.17 ❖ Выборка с недостатком с помощью связей Томека

Один из способов получить простую случайную выборку – сопоставить каждому примеру число, а затем воспользоваться генератором случайных чисел, чтобы решить, какой пример выбрать. Например, если набор данных содержит 1000 примеров, пронумерованных от 0 до 999, то для выбора примера следует использовать группы по три цифры. Так, если первые три цифры, возвращенные генератором случайных чисел, – 0, 5 и 7, то выбираем пример с номером 57 и т. д.

Простота – большой плюс этого метода выборки, его легко реализовать, потому что на любом языке программирования можно написать генератор случайных чисел. Минус же в том, что так можно не набрать достаточно примеров, обладающих конкретным свойством, представляющим интерес. Например, рассмотрим ситуацию, когда выборка производится из большого несбалансированного набора данных. При этом можно получить недостаточное количество примеров из редко встречающегося класса, а то и вообще ни одного.

3.10.2. Систематическая выборка

Чтобы реализовать **систематическую выборку** (или **интервальную выборку**), мы создаем список, содержащий все примеры. Затем мы случайным образом выбираем первый пример x_{start} из первых k элементов списка. Далее берется каждый k -й элемент списка, начиная с x_{start} . Значение k подбирается так, чтобы получить выборку нужного размера.

Преимущество систематической выборки перед простой случайной выборкой заключается в том, что примеры выбираются из всего диапазона значений. Однако систематическая выборка не годится, если список периодический или содержит повторяющиеся участки. В таком случае мы получили бы смещенную выборку. Но если список примеров рандомизирован, то систематическая выборка часто дает лучшее подмножество примеров, чем простая случайная.

3.10.3. Стратифицированная выборка

Если мы знаем о существовании в данных нескольких групп (например, по полу, местоположению или возрасту), то выборка должна включать примеры из каждой группы. Применяя **стратифицированную выборку**, мы сначала разбиваем набор данных на группы (называемые стратами), а затем случайным образом выбираем примеры из каждой страты, как при простой случайной выборке. Количество примеров, выбираемых из каждой страты, пропорционально размеру страты.

Стратифицированная выборка часто улучшает репрезентативность выборки, уменьшая смещение; в худшем случае качество результирующей выборки не хуже, чем при простой случайной выборке. Однако для определения страт аналитик должен понимать свойства набора данных. Кроме того, бывает трудно решить, какие именно атрибуты определяют страту.

Если не понятно, как лучше определить страты, то можно воспользоваться **кластеризацией**. Нужно только решить, сколько необходимо кластеров. Эта техника полезна также для выбора непомеченных примеров, которые предполагается передать человеку для пометки. Зачастую непомеченных примеров миллионы, а ресурсы для пометки ограничены. Выбирайте примеры тщательно, чтобы в помеченных данных были представлены все страты или кластеры.

Стратифицированная выборка – самый медленный из трех методов из-за дополнительных затрат на работу с несколькими независимыми стратами. Но возможность получить менее смещенную выборку часто перевешивает недостатки.

3.11. ХРАНЕНИЕ ДАННЫХ

Безопасное хранение данных – страховка бизнеса организации: если вы по какой-то причине, например из-за стихийного бедствия или человеческой

ошибки, потеряете критически важную для бизнеса модель (содержащий модель файл случайно стерт или перезаписан), то при наличии исходных данных сможете построить модель заново.

Если клиенты или партнеры предоставляют конфиденциальные данные или информацию, позволяющую установить личность (PII), то хранить их нужно не только безопасно, но и в физически защищенном месте. Совместно с администратором базы данных или инженерами DevOps вы можете ограничить доступ к конфиденциальным данным по имени пользователя или, если необходимо, по IP-адресу. Доступ к реляционной базе данных может быть также ограничен определенными строками или столбцами.

Также рекомендуется ограничить доступ только операциями чтения и добавления, а операции записи и удаления разрешать лишь отдельным пользователям.

Если данные собираются на мобильных устройствах, то, возможно, потребуется хранить их там, пока владелец устройства не подключится к сети Wi-Fi. Эти данные, возможно, необходимо шифровать, чтобы другие приложения не могли воспользоваться ими. После подключения к сети Wi-Fi данные должны быть синхронизированы с безопасным сервером по криптографически стойким протоколам, например Transport Layer Security (TLS). Каждый элемент данных на мобильном устройстве должен содержать временную метку, позволяющую правильно синхронизироваться с данными на сервере.

3.11.1. Форматы данных

Данные для машинного обучения могут храниться в разных форматах. Данные, используемые косвенно, например словари и географические справочники, можно хранить в таблице реляционной базы данных, в виде коллекции в хранилище ключей и значений или в структурированном текстовом файле.

Аккуратные данные обычно хранятся в виде последовательности значений, разделенных запятыми (CSV-файлы) или табуляторами (TSV-файлы). В таком случае все примеры хранятся в одном файле. Альтернативно можно хранить каждый пример в отдельном файле формата XML (Extensible Markup Language) или JSON (JavaScript Object Notation).

Помимо универсальных форматов, в некоторых популярных пакетах машинного обучения для хранения аккуратных данных применяются и специальные форматы. Другие пакеты нередко предоставляют интерфейсы прикладного программирования (API) к одному или нескольким специальным форматам. Чаще всего поддерживаются форматы **ARFF** (Attribute-Relation File Format, применяется в пакете машинного обучения Weka) и **LIBSVM** (Library for Support Vector Machines), которые используются по умолчанию в библиотеках LIBSVM и **LIBLINEAR** (Library for Large Linear Classification).

Данные в формате LIBSVM состоят из одного файла, содержащего все примеры. Каждая строка файла представляет помеченный вектор признаков в следующем формате:

```
label index1:value1 index2:value2 ...
```


где `indexX:valueY` описывает значение `Y` признака в позиции (измерении) `X`. Если значение в некоторой позиции равно нулю, то его можно опустить. Этот формат особенно удобен для **разреженных данных**, состоящих из примеров, в которых значения большинства признаков нулевые.

Кроме того, в разных языках программирования имеются средства **сериализации данных**. Данные для конкретного пакета машинного обучения можно сохранять на жестком диске, используя объект или функцию сериализации, предоставляемую языком либо библиотекой. При необходимости данные можно десериализовать и восстановить в исходной форме. Например, в Python имеется популярный универсальный модуль сериализации **Pickle**; в R – встроенные функции `saveRDS` и `readRDS`. Другие пакеты анализа данных могут предлагать собственные инструменты сериализации-десериализации.

В Java любой объект, реализующий интерфейс `java.io.Serializable`, можно сериализовать в файл и десериализовать, когда понадобится.

3.11.2. Уровни хранения данных

Прежде чем принимать решение, как и где хранить данные, важно выбрать подходящий **уровень хранения**. Хранение может быть организовано на разных уровнях абстракции: от самого низкого, файловой системы, до самого высокого, например озера данных.

Файловая система – это самый базовый уровень хранения. Единицей хранения на этом уровне является **файл**. Файл может быть текстовым или двоичным, его версии не контролируются, его можно легко стереть или перезаписать. Файловая система может быть локальной или сетевой. Сетевая файловая система может быть простой или распределенной.

Локальная файловая система может просто монтироваться на локальном диске и содержать все файлы, необходимые для проекта машинного обучения.

К **распределенной файловой системе**, например **NFS** (Network File System), **CephFS** (Ceph File System) или **HDFS** (Hadoop Distributed File System), можно обращаться по сети с нескольких физических или виртуальных машин. Файлы распределенной файловой системы хранятся на нескольких машинах в сети.

Несмотря на простоту, хранение на уровне файловой системы во многих случаях оправдано. Приведем несколько примеров.

Совместный доступ к файлам

Простота хранения на уровне файловой системы и поддержка стандартных протоколов позволяют с минимальными усилиями организовать совместное обращение к файлам со стороны небольшой группы сотрудников.

Локальное архивирование

Хранение на уровне файловой системы – экономичный способ архивирования данных благодаря простоте и доступности горизонтально масштабируемых решений типа NAS.

Защита данных

Хранение на уровне файловой системы можно рассматривать как решение, обеспечивающее физическую защиту данных благодаря встроенным механизмам избыточности и репликации.

Параллельный доступ к данным в файловой системе работает быстро для чтения, но медленно для записи, поэтому такой уровень хранения подходит для сравнительно небольших коллективов и данных.

Хранилище объектов – это интерфейс прикладного программирования (API), определенный поверх файловой системы. С помощью API мы можем программно выполнять такие операции над файлами, как GET, PUT и DELETE, не задумываясь о том, где на самом деле хранятся файлы. API обычно предоставляется **службой API**, расположенной в сети и доступной по протоколу **HTTP** или в более общем случае **TCP/IP** либо иному набору протоколов.

Единицей хранения в хранилище объектов является **объект**. Объекты обычно двоичные: изображения, звуковые или видеофайлы и другие элементы данных в определенном формате. Такие механизмы, как учет версий и избыточности, могут быть встроены в службу API. Доступ к данным в хранилище объектов часто можно выполнять параллельно, но не так быстро, как на уровне файловой системы.

Канонические примеры хранилищ объектов – **Amazon S3** и **Google Cloud Storage** (GCS). Альтернативная платформа хранения **Ceph** реализует хранилище объектов в одном распределенном компьютерном кластере и предоставляет интерфейсы уровня объектов и файловой системы.

Уровень **базы данных** позволяет организовать постоянное, быстрое и масштабируемое хранилище **структурированных данных** с быстрым параллельным доступом для чтения и записи.

В современной системе управления базами данных (**СУБД**, *англ.* DBMS) данные хранятся в оперативной памяти, но программное обеспечение гарантирует постоянное хранение данных (и журнала операций с данными) на диске, так что они никогда не теряются.

Единицей хранения данных на этом уровне является **строка**. У строки есть уникальный идентификатор, а значения хранятся в столбцах. В реляционной базе данных строки организованы в виде **таблиц**. Строки могут ссылаться на другие строки в той же или другой таблице.

Базы данных не очень хорошо приспособлены для хранения двоичных данных, хотя сравнительно небольшие объекты иногда можно хранить в столбце в форме **BLOB-объекта** (Binary Large Object – большой двоичный объект). BLOB представляет собой коллекцию двоичных данных, хранящуюся как единое целое. Однако чаще в строках хранятся ссылки на двоичные объекты, находящиеся где-то в другом месте – в файловой системе или в хранилище объектов.

Четыре самые популярные СУБД промышленного уровня – Oracle, MySQL, Microsoft SQL Server и PostgreSQL. Все они поддерживают SQL (Structured Query Language – структурированный язык запросов), интерфейс для доступа

и модификации данных, хранящихся в базе, а также для создания, изменения и удаления самих баз данных¹.

Озеро данных – это репозиторий данных в естественном, или первичном, формате, обычно в виде BLOB-объектов или файлов. Типичное озеро данных представляет собой неструктурированный агрегат данных из разных источников, включая базы данных, журналы или промежуточные данные, полученные в результате сложных преобразований оригинальных данных.

Данные, в т. ч. структурированные, хранятся в озере в своем естественном формате. Для чтения данных из озера аналитик должен написать программу, которая читает и разбирает файл или BLOB. Подход, предполагающий написание скрипта разбора файла или BLOB'а, называется **схемой при чтении**, в отличие от **схемы при записи**, как в СУБД. В СУБД схема данных определяется заранее и при каждой операции записи СУБД проверяет, что данные соответствуют схеме.

3.11.3. Версионирование данных

Если данные хранятся и обновляются в нескольких местах, то может возникнуть необходимость в учете версий. Версионирование необходимо также, если модель часто обновляется в результате сбора дополнительных данных, особенно автоматизированного. Так бывает, например, при разработке системы для беспилотных автомобилей, обнаружения спама или персонализированных рекомендаций. Новые данные поступают от водителя автомобиля, от пользователя, вычищающего свою электронную почту, или в результате недавнего просмотра видео.

Иногда после обновления модель начинает вести себя хуже и требуется разобраться в причинах, переключаясь с одной версии данных на другую. Версионирование данных также необходимо при обучении с учителем, когда данные помечают несколько человек. Некоторые разметчики назначают совершенно разные метки похожим примерам, что обычно снижает качество модели. Желательно хранить примеры, аннотированные разными разметчиками, отдельно и объединять их только при построении модели. Тщательный анализ качества модели может показать, что разметчики поставили плохие или несогласованные метки. Такие примеры нужно исключить из обучающих данных или разметить заново, и версионирование помогает сделать это с минимальными усилиями.

Версионирование данных можно реализовать на нескольких уровнях сложности, от самого элементарного до самого «навороченного».

Уровень 0: версионирование отсутствует

На этом уровне данные могут находиться в локальной файловой системе, в хранилище объектов или в базе данных. Преимущество неверсионированных данных – скорость и простота работы. Но это преимущество ни-

¹ В SQL Server используется проприетарный диалект Transact SQL (T-SQL), а в Oracle – Procedural Language SQL (PL/SQL).

велируется потенциальными проблемами, которые могут возникнуть при работе с моделью. Скорее всего, первой проблемой станет невозможность версионированного развертывания. В главе 8 мы будем говорить о том, что развертывание модели должно быть версионированным. Развернутая модель машинного обучения – это комбинация кода и данных. Если код версионирован, то и данные должны быть такими же. В противном случае развертывание будет неверсионированным.

А если развертывание не версионировано, то мы не сможем вернуться к прежнему уровню качества в случае возникновения проблем с моделью. Поэтому отказываться от версионирования данных не рекомендуется.

Уровень 1: версия данных – моментальный снимок на этапе обучения

На этом уровне данные версионировются путем сохранения на этапе обучения моментального снимка всего, что необходимо для обучения модели. Такой подход позволяет версионировать развернутые модели и восстанавливать предыдущее качество. Следует вести учет всех версий в каком-нибудь документе, например таблице Excel. В этом документе должны быть описаны местоположение снимка кода и данных, значения гиперпараметров и другие метаданные, нужные для воспроизведения эксперимента при необходимости. Если моделей немного и обновляются они не слишком часто, то такой уровень версионирования может оказаться приемлемым. В противном случае он не рекомендуется.

Уровень 2: данные и код версионировются совместно

На этом уровне информационные активы умеренного размера – словари, географические справочники и небольшие наборы данных – хранятся совместно с кодом в системе управления версиями, например **Git** или **Mercurial**. Большие файлы размещаются в хранилище объектов, например **S3** или **GCS**, и снабжаются уникальными идентификаторами. Обучающие данные хранятся в стандартном формате, например JSON, XML или еще каком-то, и включают релевантные метаданные, например метки, идентификатор разметки, время разметки, инструмент разметки и т. д.

Инструменты типа **Git Large File Storage (LFS)** автоматически заменяют большие файлы, например звуковые дорожки, видео, большие наборы данных и графику, текстовые указатели, а содержимое файлов хранят на удаленном сервере.

Версия набора данных определяется **git-сигнатурами** файла кода и данных. Также полезно добавить временную метку, чтобы упростить идентификацию конкретной версии.

Уровень 3: использование или создание специализированного решения по версионированию данных

Такие системы контроля версий, как **DVC** и **Pachyderm**, предлагают дополнительные инструменты версионирования данных. Обычно они умеют взаимодействовать с системами контроля версий кода типа **Git**.

Версионирование уровня 2 рекомендуется для большинства проектов. Если вы полагаете, что для ваших нужд уровня 2 недостаточно, изучите ре-

шения уровня 3 или создайте свое собственное. В других случаях этот путь не рекомендуется, поскольку увеличивает сложность и без того сложного технического проекта.

3.11.4. Документация и метаданные

Во время активной работы над проектом машинного обучения мы обычно помним важные сведения, относящиеся к данным. Но после сдачи проекта в эксплуатацию мы переключаемся на другой проект, и эта информация постепенно забывается.

Прежде чем заняться другим проектом, позаботьтесь о том, чтобы другие люди могли разобраться в ваших данных и правильно использовать их.

Если данные не требуют специальных пояснений, можете не документировать их. Но вообще редко бывает так, чтобы человек, не создававший набор данных, мог легко разобраться в нем и понять, как его использовать, просто глядя на данные.

Документация должна сопровождать любые информационные активы, использованные при обучении модели. В документации должны быть отражены следующие детали:

- что означают данные;
- как они собирались или какими методами создавались (инструкции для разметчиков и способы контроля качества);
- как производилось разбиение на обучающий, контрольный и тестовый наборы;
- подробное описание всех шагов предобработки;
- по каким причинам данные исключались;
- какой формат используется для хранения данных;
- типы атрибутов или признаков (какие значения может принимать каждый атрибут или признак);
- количество примеров;
- возможные значения меток или допустимый диапазон числового целевого показателя.

3.11.5. Жизненный цикл данных

Некоторые данные могут храниться неопределенно долго. Но в деловой практике бывает, что данные разрешено хранить в течение некоторого времени, а затем необходимо удалить. Если к данным, с которыми вы работаете, применимы такие ограничения, то обязательно следует реализовать надежный механизм оповещения, который будет контактировать с человеком, ответственным за стирание данных, и иметь резервный план на случай, если этот человек недоступен. Не забывайте, что иногда за несвоевременное стирание данных предусмотрена серьезная ответственность.

Для любого конфиденциального информационного актива должен существовать **документ о жизненном цикле данных**, в котором описан актив

и круг людей, имеющих к нему доступ как во время разработки, так и после завершения проекта. В документе должно быть указано, сколько времени должен храниться актив и следует ли его явно уничтожать.

3.12. ДОПОЛНИТЕЛЬНЫЕ РЕКОМЕНДАЦИИ ПО РАБОТЕ С ДАННЫМИ

В заключение этой главы рассмотрим еще две рекомендации: воспроизводимость и принцип «сначала данные, потом алгоритм».

3.12.1. Воспроизводимость

Воспроизводимость должна учитываться со всей серьезностью, что бы вы ни делали, включая сбор и подготовку данных. Избегайте преобразования данных вручную, а также использования мощных средств, встроенных в текстовые редакторы и командные оболочки, как то: регулярные выражения, написанные «по случаю» команды `awk` или `sed`, конвейеры команд.

Обычно операции сбора и преобразования данных состоят из нескольких этапов. Сюда входит загрузка данных из веба или баз данных, замена многословных выражений уникальными токенами, удаление стоп-слов и шума, кадрирование и повышение резкости изображений, подстановка отсутствующих значений и т. д. Каждый этап процесса должен быть реализован в виде программного скрипта, например на языке Python или R, имеющего входные параметры и результат на выходе. Если ваша работа организована именно так, то вы сможете отследить любые изменения в данных. Если на каком-то этапе с данными случится неприятность, вы всегда сможете исправить скрипт и запустить весь конвейер обработки сначала.

С другой стороны, ручное вмешательство воспроизвести трудно. Такие действия бывает нелегко применить к обновленным данным или масштабировать на гораздо большие объемы данных (когда у вас появится возможность получить больше данных или другой набор данных).

3.12.2. Сначала данные, потом алгоритм

Помните, что в индустрии, в отличие от академических кругов, действует принцип «сначала данные, потом алгоритм», поэтому тратьте время и силы на получение как можно большего количества разнообразных высококачественных данных, а не на попытки выжать максимум производительности из алгоритма обучения.

Приращение данных, если оно хорошо реализовано, лучше послужит качеству модели, чем поиск оптимальных значений гиперпараметров или архитектуры модели.

3.13. РЕЗЮМЕ

Прежде чем приступить к сбору данных, ответьте на пять вопросов: являются ли данные, с которыми вы собираетесь работать, доступными, объемными, пригодными для использования, понятными и надежными.

Типичные проблемы с данными – высокая стоимость, смещение, низкая предсказательная способность, устаревшие примеры и просачивание.

Хорошие данные содержат достаточно информации, которую можно использовать для моделирования, в полной мере покрывают все, что вы хотите делать с моделью, и отражают реальные входы, которые модель будет получать в производственном режиме. Они являются настолько несмещенными, насколько это возможно, и не являются результатом самой модели, имеют согласованные метки и достаточно велики для обобщаемости модели.

Хорошие данные о взаимодействии с пользователем содержат информацию о трех аспектах: контекст взаимодействия, действие пользователя в этом контексте и результат взаимодействия.

Для получения хорошего разбиения всего набора данных на обучающий, контрольный и тестовый наборы процесс разбиения должен удовлетворять нескольким условиям: 1) данные были рандомизированы до разбиения; 2) разбиение применялось к первичным данным; 3) распределение данных в контрольном и тестовом наборах одинаково; 4) было предотвращено просачивание.

Методы подстановки данных можно использовать для обработки отсутствия некоторых атрибутов.

Методы приращения данных часто используются для получения новых помеченных примеров без дополнительной ручной разметки. Эти методы обычно применяются к изображениям, но могут также применяться к тексту и другим типам данных, связанных с восприятием.

Несбалансированность классов может оказать существенное влияние на качество модели. Алгоритмы обучения работают неоптимально, если обучающие данные не сбалансированы. Такие методы, как выборка с избытком и с недостатком, помогают решить проблему несбалансированности классов.

Имея дело с большими данными, не всегда разумно и необходимо обрабатывать весь набор данных. Вместо этого произведите сравнительно небольшую выборку данных, содержащую достаточно информации для обучения. Можно использовать разные стратегии выборки, в т. ч. простую случайную выборку, систематическую выборку, стратифицированную выборку и кластерную выборку.

Данные можно хранить в разных форматах и на разных уровнях. Версионирование данных – обязательный элемент обучения с учителем, если разметка производится несколькими людьми. Разные разметчики могут ставить метки разного качества, поэтому важно следить за тем, кто какой помеченный пример создал. Версионирование данных можно реализовать по-разному: от самого простого до самого «навороченного» способа; вообще не контролировать версии (уровень 0), версия данных – моментальный снимок на этапе обучения (уровень 1), версионировать в виде одного актива, содер-

жащего данные и код (уровень 2), использовать или создать самостоятельно специализированное решение для версионирования данных (уровень 3).

Для большинства проектов рекомендуется уровень 2.

Любой информационный актив, использованный при обучении модели, должен быть документирован. В документации должны быть отражены следующие детали: что означают данные, как они были собраны или как создавались (инструкции для разметчиков и методы контроля качества), детали разбиения на обучающий, контрольный и тестовый наборы и всех шагов предобработки. Обязательно должны быть включены следующие пояснения: какие данные были исключены, в каком формате хранятся данные, типы атрибутов или признаков, количество примеров и возможные значения меток или допустимые диапазоны числовых целевых показателей.

Для каждого конфиденциального информационного актива в документе о жизненном цикле данных должен быть описан актив и круг лиц, имеющих к нему доступ во время работы над проектом и после ее завершения.

Глава 4

Конструирование признаков

Конструирование признаков – следующая ступень после сбора и подготовки данных из важнейших частей машинного обучения. Это также третий этап жизненного цикла проекта машинного обучения.

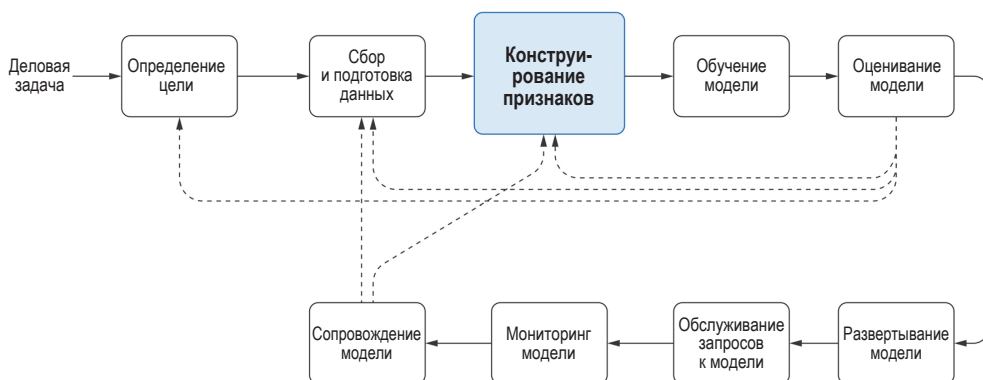


Рис. 4.1 ❖ Жизненный цикл проекта машинного обучения

Смысл конструирования признаков заключается в том, чтобы сначала на концептуальном уровне, а затем программно преобразовать первичный пример в вектор признаков. Требуется написать код, который преобразует первичный пример в признак, возможно, с привлечением каких-то косвенных данных.

4.1. ЗАЧЕМ КОНСТРУИРОВАТЬ ПРИЗНАКИ

Для конкретики рассмотрим задачу распознавания названий кинофильмов в твитах. Допустим, что имеется обширная коллекция названий фильмов;

эти данные мы будем использовать **косвенно**. Также имеется коллекция твитов, они будут использоваться **напрямую** для создания примеров. Сначала построим индекс названий фильмов с целью быстрого поиска строк¹. Теперь условимся, что наши примеры – это найденные соответствия, а задача машинного обучения заключается в бинарной классификации: является соответствие фильмом или не является.

Рассмотрим следующий твит:

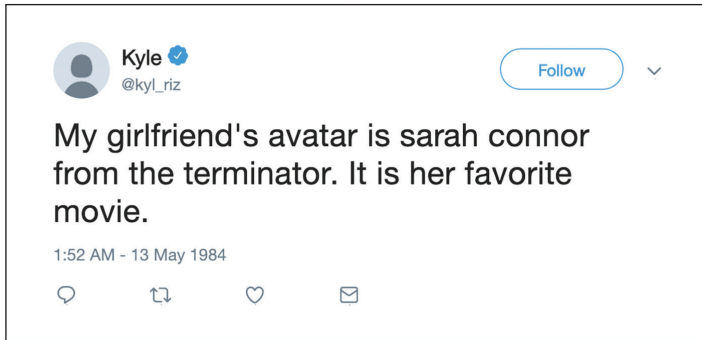


Рис. 4.2 ❖ Твит от Кайла

Наш индекс названий фильмов поможет найти следующие соответствия: «avatar», «the terminator», «It» и «her». Это дает нам четыре непомеченных примера. Мы можем пометить их: {(avatar, False), (the terminator, True), (It, False), (her, False)}. Однако алгоритм машинного обучения не сможет ничему обучиться на одних лишь названиях фильмов (как, впрочем, не смог бы и человек), необходим контекст. Допустим, что пять предшествующих и пять последующих слов образуют достаточно информативный контекст. В терминологии машинного обучения такой контекст называется «10-словным окном» вокруг найденной строки. Ширину окна можно считать гиперпараметром и настроить.

Итак, примерами являются помеченные соответствия в контексте. Однако к таким данным алгоритм обучения неприменим. Алгоритмы обучения применимы только к векторам признаков. Тут-то и наступает черед конструирования признаков.

4.2. КАК КОНСТРУИРУЮТСЯ ПРИЗНАКИ

Конструирование признаков – это творческий процесс, в ходе которого аналитик использует свое воображение, интуицию и знания предметной области. В примере распознавания названий фильмов в твитах мы воспользовались интуицией при задании ширины контекстного окна. А теперь нужно

¹ Построить индекс для быстрого поиска строк можно, например, с помощью **алгоритма Ахо–Корасик**.

проявить еще больше изобретательности, чтобы преобразовать последовательности строк в числовые векторы.

4.2.1. Конструирование признаков для текста

При обработке текста ученые и инженеры часто применяют простые приемы конструирования признаков. Два из них называются унитарным кодированием и мешком слов.

В общем случае **унитарное кодирование** преобразует категориальный атрибут в несколько бинарных. Допустим, что в наборе данных имеется атрибут «Color» (Цвет), принимающий значения «red», «yellow» и «green». Преобразуем каждое значение в трехмерный бинарный вектор:

```
red = [1, 0, 0];
yellow = [0, 1, 0];
green = [0, 0, 1].
```

В электронной таблице вместо одного столбца «Color» мы заведем три синтетических столбца со значениями 1 или 0. Преимущество в том, что категориальные атрибуты поддерживают лишь очень немногие алгоритмы машинного обучения, а теперь в нашем распоряжении гораздо более широкий спектр алгоритмов.

Мешок слов – это обобщение техники унитарного кодирования на текстовые данные. Вместо того чтобы представлять бинарным вектором один атрибут, мы воспользуемся этой техникой для представления всего текстового документа в виде бинарного вектора. Посмотрим, как это работает.

Рассмотрим набор из шести текстовых документов:

Документ 1	Love, love is a verb
Документ 2	Love is a doing word
Документ 3	Feathers on my breath
Документ 4	Gentle impulsion
Документ 5	Shakes me, makes me lighter
Документ 6	Feathers on my breath

Рис. 4.3 ❖ Набор из шести документов

Предположим, что задача заключается в построении классификатора текста по теме. Алгоритм обучения классификатора ожидает, что на вход будут поданы помеченные векторы признаков, поэтому нужно преобразовать набор текстовых документов в набор векторов признаков. Именно для этого и предназначен мешок слов.

Прежде всего текст подвергается лексемизации. **Лексемизация** – это процедура разбиения текста на фрагменты, называемые «лексемами». **Лексемизатором** называется программа, которая принимает на входе строку и воз-

вращает последовательность извлеченных из нее лексем. Чаще всего лексемы представляют собой слова, но это необязательно. Лексемой может быть знак препинания, слово, а иногда словосочетание, например название компании (McDonald's) или места (Красная площадь). Мы будем использовать лексемизатор, который извлекает слова и игнорирует все остальное. Тогда получится следующий набор:

Документ 1	[Love, love, is a verb]
Документ 2	[Love, is, a, doing, word]
Документ 3	[Feathers, on, my, breath]
Документ 4	[Gentle, impulsions]
Документ 5	[Shakes, me, makes, me lighter]
Документ 6	[Feathers, on, my, breath]

Рис. 4.4 ❖ Набор лексемизированных документов

Следующий шаг – построение словаря. Он содержит 16 лексем¹:

a	breath	doing	feathers
gentle	impulsion	is	lighter
love	makes	me	my
on	shakes	verb	word

Теперь каким-то образом упорядочим этот словарь и назначим уникальный индекс каждой лексеме. Я решил упорядочить по алфавиту.

a	breath	doing	feathers	gentle	impulsion	is	lighter	love	makes	me	my	on	shakes	verb	word
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Рис. 4.5 ❖ Упорядоченные и проиндексированные лексемы

У каждой лексемы в словаре имеется уникальный индекс – число от 1 до 16. Преобразуем этот набор в набор бинарных векторов признаков, как показано ниже на рис. 4.6.

Наличие 1 в некоторой позиции означает, что соответствующая лексема встречается в тексте. В противном случае в этой позиции вектора находится 0.

Например, документ 1 «Love, love is a verb» представлен таким вектором признаков:

[1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0]

¹ Я решил игнорировать прописные буквы, но вообще-то аналитик может рассматривать лексемы «Love» и «love» как два разных словарных слова.

	word															
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Документ 1	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0
Документ 2	1	0	1	0	0	0	1	0	1	0	0	0	0	0	0	1
Документ 3	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0
Документ 4	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
Документ 5	0	0	0	0	0	0	0	1	0	1	1	0	0	1	0	0
Документ 6	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0

Рис. 4.6 ❖ Векторы признаков

Будем использовать соответствующие помеченные векторы признаков как обучающие данные, с которыми может работать любой алгоритм классификации.

Есть несколько разновидностей мешка слов. Описанная выше модель бинарных значений зачастую дает хорошие результаты. Но у бинарных значений есть альтернативы: 1) счетчики лексем, 2) частоты лексем, 3) **TF-IDF** (частота термина – обратная частота документа). Если использовать счетчики лексем, то значением признака для лексемы «love» в документе 1 «Love, love is a verb» будет 2 – столько раз она встречается в этом документе. Если использовать частоты лексем, то значением для «love» будет $2/5 = 0.4$ в предположении, что лексемизатор выделил две лексемы «love», а всего в документе 1 пять лексем. Значение TF-IDF пропорционально частоте слова в документе и обратно пропорционально количеству документов в корпусе, содержащих это слово. Для некоторых слов оно корректируется, например для предлогов и местоимений, которые, вообще говоря, встречаются часто. Я не буду вдаваться в детали TF-IDF, а порекомендую интересующимся читателям найти дополнительную информацию в интернете.

Прямолинейное обобщение мешка слов – **мешок n -грамм**. **n -граммой** называется последовательность n слов в корпусе. Если $n = 2$ и знаки препинания игнорируются, то в тексте «No, I am your father» присутствуют следующие 2-граммы (обычно они называются **биграммами**): [«No I», «I am», «am your», «your father»]. А вот полный список триграмм: [«No I am», «I am your», «am your father»]. Объединяя все n -граммы вплоть до некоторого n вместе с лексемами в одном словаре, мы получаем мешок n -грамм, который можно лексемизировать так же, как мы поступали в модели мешка слов.

Поскольку последовательности слов обычно встречаются реже, чем отдельные слова, использование n -грамм приводит к созданию более **разреженного** вектора признаков. В то же время n -граммы позволяют алгоритму машинного обучения обучить модель, учитывающую больше нюансов. Например, выражения «this movie was not good and boring» (этот фильм был нехорошим и скучным) и «this movie was good and not boring» (этот фильм был хорошим и нескучным) имеют противоположный смысл, но мешок слов для них одинаков, поскольку состоит только из самих слов. Если же рассматривать биграммы слов, то векторы биграмм в мешке слов будут различны.

4.2.2. Почему мешок слов работает

Векторы признаков работают только при соблюдении определенных правил. Одно из них – признак в позиции j вектора признаков должен представлять одно и то же свойство для всех примеров в наборе данных. Если для какого-то человека в наборе данных признак представляет рост в сантиметрах, то и для всех остальных примеров это утверждение должно быть верным. Признак в позиции j должен всегда представлять рост в сантиметрах и ничто иное.

Метод мешка слов удовлетворяет этому условию. Каждый признак представляет одно и то же свойство документа: присутствует некая лексема в документе или отсутствует.

Другое правило требует, чтобы похожие векторы признаков представляли похожие сущности в наборе данных. Это правило также выполняется для мешка слов. У двух одинаковых документов будут одинаковые векторы признаков. Аналогично для двух текстов на одну и ту же тему векторы признаков с большой вероятностью будут похожи, потому что в них больше общих слов, чем в документах на разные темы.

4.2.3. Преобразование категориальных признаков в числа

Унитарное кодирование – не единственный способ преобразовать категориальные признаки в числа и не всегда наилучший.

Другой метод – **кодирование средним**, называемое также **подсчетом интервалов** или **калибровкой признаков**. Сначала по всем примерам, в которых признак имеет значение z , вычисляется **выборочное среднее** меток. Затем каждое значение z категориального признака заменяется этим средним. Преимущество этого метода в том, что размерность данных не увеличивается, по построению, числовое значение несет некоторую информацию о метке.

Если мы решаем задачу бинарной классификации, то в дополнение к выборочному среднему можно использовать другие полезные величины: сами счетчики положительного класса для данного значения z , **отношение шансов** и **логарифм отношения шансов**. Отношение шансов (odds ratio – OR) обычно определяется для двух случайных величин. В общем случае OR – это статистика, которая количественно выражает силу связи между двумя событиями A и B . Два события считаются независимыми, если OR равно 1, т. е. шансы одного события одинаковы вне зависимости от того, произошло или не произошло другое событие.

Что касается количественного выражения категориального признака, мы можем вычислить отношение шансов между значением z категориального признака (событие A) и положительной меткой (событие B). Проиллюстрируем на примере. Пусть требуется предсказать, является почтовое сообщение спамом или нет. Предположим, что имеется размеченный набор почтовых сообщений, и мы сконструировали признак, содержащий самое частое слово

в каждом сообщении. Найдем числовое значение, которое могло бы заменить категориальное значение «infected» этого признака. Сначала построим **таблицу сопряженности** слова «infected» и метки «spam»:

	Спам	Не спам	Всего
Содержит «infected»	145	8	153
Не содержит «infected»	346	2909	3255
Всего	491	2917	3408

Рис. 4.7 ❖ Таблица сопряженности для «infected» и «spam»

Отношение шансов «infected» и «spam» равно

$$\text{odds ratio}(\text{infected}, \text{spam}) = \frac{145/8}{346/2909} = 152.4.$$

Как видим, в зависимости от значений в таблице сопряженности отношение шансов может быть как очень малым (близким к нулю), так и очень большим (сколь угодно большим положительным числом). Чтобы избежать численного переполнения, часто применяется логарифм отношения шансов:

$$\begin{aligned} \log \text{odds ratio}(\text{infected}, \text{spam}) &= \log(145/8) - \log(346/2909) \\ &= \log(145) - \log(8) - \log(346) + \log(2909) = 2.2. \end{aligned}$$

Теперь можно заменить значение «infected» в рассматриваемом категориальном признаке числом 2.2. Точно так же можно поступить для других значений категориального признака, преобразовав их в логарифм отношения шансов.

Иногда категориальные признаки упорядочены, но не являются циклическими. Примерами могут служить школьные отметки (от «А» до «Е») и уровни квалификации («младший», «средний», «старший»). Вместо унитарного кодирования удобно представлять их осмысленными числами. Например, можно взять равноотстоящие числа в диапазоне $[0, 1]$: $1/3$ для «младший», $2/3$ для «средний» и 1 для «старший». Если расстояния между значениями не должны быть равны, это можно отразить в выборе отношений. Так, если «старший» должно отстоять от «средний» дальше, чем «средний» отстоит от «младший», то можно было бы представить эти уровни числами $1/5$, $2/5$ и 1 . Поэтому так важно разбираться в предметной области.

Если категориальные признаки циклические, то кодирование целыми числами не годится. Например, попробуем преобразовать дни недели от понедельника до воскресенья числами от 1 до 7. Тогда разность между субботой и воскресеньем равна 1, а между воскресеньем и понедельником –6. Однако было бы естественно предположить, что разность должна быть одинаковой и равной 1, потому что понедельник следует сразу за воскресеньем.

Вместо этого воспользуемся **синусно-косинусным преобразованием**. Оно преобразует циклический признак в два синтетических. Обозначим p

целочисленное значение нашего циклического признака. Заменим p двумя значениями:

$$p_{\sin} = \sin\left(\frac{2 \times \pi \times p}{\max(p)}\right), \quad p_{\cos} = \cos\left(\frac{2 \times \pi \times p}{\max(p)}\right).$$

В таблице ниже приведены значения p_{\sin} и p_{\cos} для всех семи дней недели:

p	p_{\sin}	p_{\cos}
1	0.78	0.62
2	0.97	-0.22
3	0.43	-0.9
4	-0.43	-0.9
5	-0.97	-0.22
6	-0.78	0.62
7	0	1

На рис. 4.8 показана диаграмма рассеяния для этой таблицы. Видна циклическая природа обоих новых признаков.

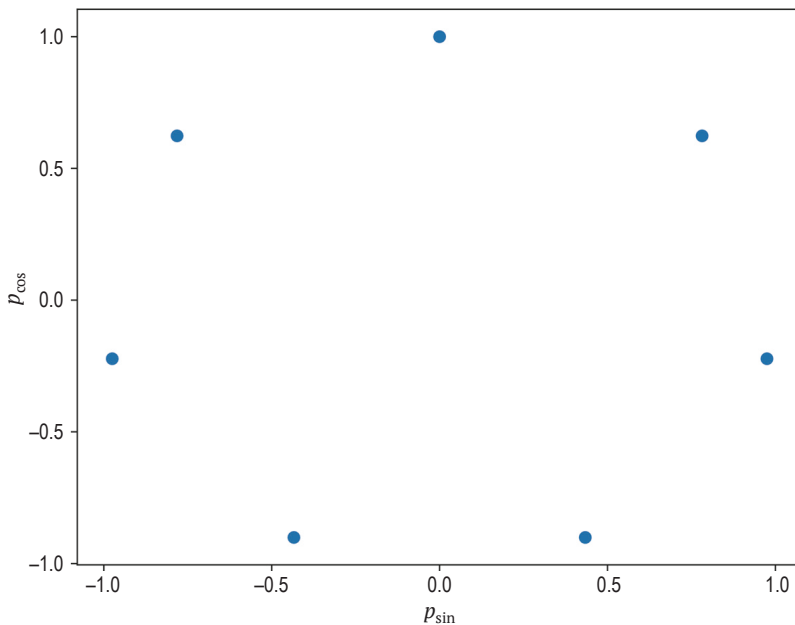


Рис. 4.8 ❖ Признак, представляющий дни недели, после синусно-косинусного преобразования

Теперь в аккуратных данных заменим «понедельник» двумя значениями $[0.78, 0.62]$, «вторник» – $[0.97, -0.22]$ и т. д. В набор данных добавилось одно измерение, но прогностическое качество модели значительно улучшилось по сравнению с целочисленным кодированием.

4.2.4. Хеширование признаков

Хеширование признаков позволяет преобразовать текстовые данные или категориальные атрибуты, принимающие много значений, в вектор признаков произвольной размерности. У унитарного кодирования и мешка слов есть недостаток: если уникальных значений много, то векторы признаков будут иметь высокую размерность. Например, если в коллекции текстовых документов имеется миллион уникальных лексем, то модель мешка слов породит векторы признаков размерности миллион. Работа с данными такой большой размерности обходится очень дорого в вычислительном плане.

Чтобы не запутаться в данных, хеширование можно применять следующим образом. Сначала решите, какой должна быть размерность векторов признаков. Затем с помощью **функции хеширования** преобразуйте все значения категориального атрибута (или все лексемы в коллекции документов) в числа, которые станут индексами векторов признаков. Эта процедура показана на рис. 4.9.

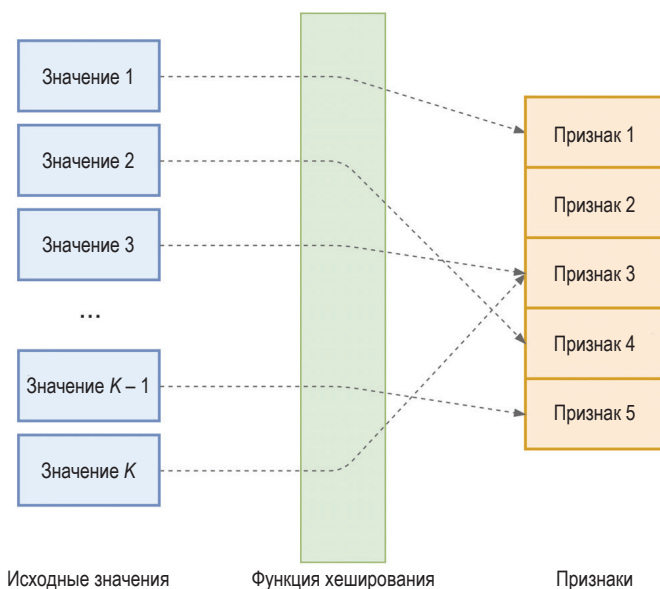


Рис. 4.9 ❖ Хеширование признаков для получения желаемой размерности 5, когда первоначально атрибут имеет K значений

Проиллюстрируем, как это работает, на примере преобразования текста «Love is a doing word» в вектор признаков. Пусть имеется функция хеширования h , которая принимает на входе строку и выводит неотрицательное целое число. Будем считать, что желаемая размерность равна 5. Применив функцию хеширования к каждому слову и взяв остаток от деления результата на 5 в качестве индекса слова, получим:

$h(\text{love}) \bmod 5 = 0$
 $h(\text{is}) \bmod 5 = 3$
 $h(\text{a}) \bmod 5 = 1$
 $h(\text{doing}) \bmod 5 = 3$
 $h(\text{word}) \bmod 5 = 4$

Теперь построим вектор признаков:

[1, 1, 0, 2, 1].

Действительно, $h(\text{love}) \bmod 5 = 0$ означает, что по измерению 0 вектора признаков имеется одно слово; $h(\text{is}) \bmod 5 = 3$ и $h(\text{doing}) \bmod 5 = 3$ – что по измерению 3 имеется два слова и т. д. Как легко видеть, между словами «is» и «doing» имеется **коллизия**: оба представлены измерением 3. Чем меньше желаемая размерность, тем больше шансы получить коллизию. Это компромисс между скоростью и качеством обучения.

Для хеширования часто используются функции **MurmurHash3**, **Jenkins**, **CityHash** и **MD5**.

4.2.5. Тематическое моделирование

Тематическое моделирование – это семейство методов, в которых используются непомеченные данные, обычно в форме текстовых документов на естественном языке. Модель обучается представлять документ в виде вектора тем. Например, в коллекции новостей первые пять тем – «спорт», «политика», «развлечения», «финансы» и «технологии». Поэтому каждый документ можно было бы представить пятимерным вектором признаков, по одному измерению на тему:

[0.04, 0.5, 0.1, 0.3, 0.06]

Этот конкретный вектор представляет документ, отнесенный к двум главным темам: политика (с весом 0.5) и финансы (с весом 0.3). Алгоритмы тематического моделирования, в частности **латентный семантический анализ** (Latent Semantic Analysis – LSA) и **латентное распределение Дирихле** (Latent Dirichlet Allocation – LDA), обучаются путем анализа непомеченных документов. Эти два алгоритма порождают похожие выходы, но основаны на разных математических моделях. В LSA используется **сингулярное разложение** (singular value decomposition – SVD) матрицы соответствия слов и документов (которая строится с помощью **мешка слов** или **TF-IDF**), а в LDA – иерархическая **байесовская модель**, в которой каждый документ является **смесью** нескольких тем, а присутствие каждого слова объясняется одной из тем.

Проиллюстрируем, как это работает. Ниже приведен код LSA на Python:

```

1 from sklearn.feature_extraction.text import TfidfVectorizer
2 from sklearn.decomposition import TruncatedSVD
3
```

```

4 class LSA():
5     def __init__(self, docs):
6         # Преобразовать документ в векторы TF-IDF
7         self.TF_IDF = TfidfVectorizer()
8         self.TF_IDF.fit(docs)
9         vectors = self.TF_IDF.transform(docs)
10
11        # Построить тематическую модель LSA
12        self.LSA_model = TruncatedSVD(n_components=50)
13        self.LSA_model.fit(vectors)
14        return
15
16    def get_features(self, new_docs):
17        # Получить тематические признаки для новых документов
18        new_vectors = self.TF_IDF.transform(new_docs)
19        return self.LSA_model.transform(new_vectors)
20
21    # Позже, на этапе эксплуатации, создать экземпляр модели LSA
22    docs = ["This is a text.", "This another one."]
23    LSA_featurizer = LSA(docs)
24
25    # Получить тематические признаки для new_docs
26    new_docs = ["This is a third text.", "This is a fourth one."]
27    LSA_features = LSA_featurizer.get_features(new_docs)

```

Соответствующий код¹ на R показан ниже:

```

1 library(tm)
2 library(lsa)
3
4 get_features <- function(LSA_model, new_docs){
5     # new_docs можно передать как объект tm::Corpus или как вектор,
6     # сохраняя символьные строки, представляющие документы
7     if(!inherits(new_docs, "Corpus"))
8         new_docs <- VCorpus(VectorSource(new_docs))
9     tdm_test <- TermDocumentMatrix(
10        new_docs,
11        control = list(
12            dictionary = rownames(LSA_model$tk),
13            weighting = weightTfIdf
14        )
15    )
16    txt_mat <- as.textmatrix(as.matrix(tdm_test))
17    crossprod(t(crossprod(txt_mat, LSA_model$tk)), diag(1/LSA_model$sk))
18 }
19 # Обучить модель LSA на корпусе docs
20 docs <- c("This is a text.", "This another one.")
21 corpus <- VCorpus(VectorSource(docs))
22 tdm_train <- TermDocumentMatrix(

```

¹ Код на R для LSA и LDA предоставил Джулиан Эймон.

```

23 corpus, control = list(weighting = weightTfIdf))
24 txt_mat <- as.textmatrix(as.matrix(tdm_train))
25 LSA_fit <- lsa(txt_mat, dims = 2)
26
27 # Позже, на этапе эксплуатации, получить тематические признаки для
# new_docs
28 new_docs <- c("This is a third text.", "This is a fourth one.")
29 LSA_features <- get_features(LSA_fit, new_docs)

```

Ниже приведен код LDA на Python:

```

1 from sklearn.feature_extraction.text import CountVectorizer
2 from sklearn.decomposition import LatentDirichletAllocation
3
4 class LDA():
5     def __init__(self, docs):
6         # Преобразовать документы в векторы TF-IDF
7         self.TF = CountVectorizer()
8         self.TF.fit(docs)
9         vectors = self.TF.transform(docs)
10        # Построить тематическую модель LDA
11        self.LDA_model = LatentDirichletAllocation(n_components=50)
12        self.LDA_model.fit(vectors)
13        return
14    def get_features(self, new_docs):
15        # Получить тематические признаки для новых документов
16        new_vectors = self.TF.transform(new_docs)
17        return self.LDA_model.transform(new_vectors)
18
19 # Позже, на этапе эксплуатации, создать экземпляр модели LDA
20 docs = ["This is a text.", "This another one."]
21 LDA_featurizer = LDA(docs)
22
23 # Получить тематические признаки для new_docs
24 new_docs = ["This is a third text.", "This is a fourth one."]
25 LDA_features = LDA_featurizer.get_features(new_docs)

```

А вот соответствующий код на R:

```

1 library(tm)
2 library(topicmodels)
3
4 # Сгенерировать признаки для new_docs, используя модель LDA_model
5 get_features <- function(LDA_mode, new_docs){
6     # new_docs можно передать как объект tm::Corpus или как вектор,
7     # сохраняя символьные строки, представляющие документы
8     if(!inherits(new_docs, "Corpus"))
9         new_docs <- VCorpus(VectorSource(new_docs))
10    new_dtm <- DocumentTermMatrix(new_docs, control = list(weighting =
11        weightTf))
12    posterior(LDA_mode, newdata = new_dtm)$topics
13 }

```

```

13 # обучить модель LDA на корпусе docs
14 docs <- c("This is a text.", "This another one.")
15 corpus <- VCorpus(VectorSource(docs))
16 dtm <- DocumentTermMatrix(corpus, control = list(weighting = weightTf))
17 LDA_fit <- LDA(dtm, k = 5)
18
19 # Позже, на этапе эксплуатации, получить тематические признаки для
20 # new_docs
21 new_docs <- c("This is a third text.", "This is a fourth one.")
22 LDA_features <- get_features(LDA_fit, new_docs)

```

В этих листингах `docs` – коллекция текстовых документов. Это может быть, например, список строк, в котором каждая строка является документом.

4.2.6. Признаки для временных рядов

Временные ряды отличаются от традиционных данных для обучения с учителем, которые имеют вид неупорядоченной коллекции независимых наблюдений. Временной ряд – это упорядоченная коллекция наблюдений, каждый элемент которой содержит какой-то связанный со временем атрибут, например временную метку, дату, год и месяц, год и т. д. На рис. 4.10 приведен пример временного ряда.

Date	Stock Price	S&P 500	Dow Jones
2020-01-11
2020-01-12	14.5	3345	28 583
2020-01-12	14.7	3352	28 611
2020-01-12	15.9	3347	29 001
2020-01-13	17.9	3298	28 312
2016-01-13	16.8	3521	28 127
2020-01-14	17.9	3687	28 564
2016-01-15	16.8	3540	27 998
2016-01-16

Рис. 4.10 ❖ Пример временного ряда в виде потока событий

На рис. 4.10 каждая строка содержит стоимость некоторой акции в определенный момент времени, а также два индекса: S&P 500 и Dow Jones. Наблюдения производились нерегулярно: 2020-01-12 было произведено три наблюдения, а 2020-01-13 – только два. В **классических временных рядах** наблюдения равноотстоят во времени, например одно наблюдение в секунду, в минуту, в сутки и т. д. Если наблюдения нерегулярны, то данные называются **точечным процессом**, или **потокком событий**.

Обычно поток событий можно преобразовать в классический временной ряд, агрегировав наблюдения. Примерами операторов агрегирования явля-

ются COUNT и AVERAGE. Применяв оператор AVERAGE к потоку событий на рис. 4.10, мы получим классический временной ряд, показанный на рис. 4.11.

Date	Stock Price	S&P 500	Dow Jones
2020-01-11
2020-01-12	15.0	3348	28 732
2020-01-13	17.4	3410	28 220
2020-01-14	17.9	3687	28 564
2016-01-15	16.8	3540	27 998
2016-01-16

Рис. 4.11 ❖ Классический временной ряд, полученный агрегированием потока событий на рис. 4.10

Хотя с потоками событий можно работать непосредственно, приведение временного ряда к классическому виду упрощает последующее агрегирование и генерирование признаков для обучения.

Аналитики чаще всего используют временные ряды для решения двух видов задач предсказания. Пусть дана последовательность недавних наблюдений. Требуется:

- предсказать что-то, относящееся к следующему наблюдению (например, зная цену акции и значения биржевых индексов за последние семь дней, предсказать цену акции на завтра);
- предсказать что-то о явлении, породившем эту последовательность (например, по журналу подключений пользователя к программной системе предсказать, отменит ли он подписку в текущем квартале).

До того как нейронные сети достигли современной способности к обучению, аналитики работали с временными рядами, используя средства **поверхностного машинного обучения**. Чтобы преобразовать временной ряд в обучающие данные в форме векторов признаков, необходимо принять два решения:

- сколько последовательных наблюдений необходимо для точного предсказания (так называемое окно предсказания);
- как преобразовать последовательность наблюдений в вектор признаков фиксированной размерности.

Ни на один из этих вопросов нет простого ответа. Обычно решение опирается на знания эксперта о предметной области или на основе **настройки гиперпараметров**. Однако есть несколько рецептов, которые хорошо работают для многих временных рядов. Вот один из них.

- 1) разбить временной ряд на сегменты длины w ;
- 2) создать обучающий пример e для каждого сегмента s ;
- 3) для каждого e вычислить различные статистики по наблюдениям в s .

Возьмем данные на рис. 4.11 и разобьем их на сегменты длины $w = 2$, где w – длина окна предсказания. На рис. 4.12 показано, что каждый сегмент теперь является отдельным примером.

пример i				пример $i + 1$			
$t - 2$	15.0	3348	28 732	$t - 2$	17.4	3410	28 220
$t - 1$	17.4	3410	28 220	$t - 1$	17.9	3687	28 564

пример $i + 2$			
$t - 2$	17.9	3687	28 564
$t - 1$	16.8	3540	27 998

Рис. 4.12 ❖ Временной ряд, разбитый на сегменты длиной $w = 2$

На практике w обычно больше 2. Допустим, что окно предсказания имеет длину 7. Статистика, вычисляемая на шаге 3, может быть одной из следующих:

- среднее (например, **среднее арифметическое** или **медиана** цен акции за последние семь дней);
- разброс (например, **стандартное отклонение**, **медианное абсолютное отклонение** или **межквартильный размах** значений индекса S&P 500 за последние семь дней);
- выбросы (например, доля наблюдений, в которых индекс Доу–Джонса был аномально малым, скажем более чем два стандартных отклонения от среднего);
- рост (например, выросло ли значение индекса S&P 500 между днем $t - 6$ и t , днем $t - 3$ и t и днем $t - 1$ и t);
- визуальные образы (например, насколько кривая цены акции отличается от известного визуального образа, скажем шляпы или головы с плечами).

Теперь вы понимаете, почему рекомендуется преобразовывать временной ряд в классическую форму: все перечисленные выше статистики имеют смысл только при вычислении на сравнимых значениях.

Следует отметить, что в современную эпоху нейронных сетей аналитики чаще предпочитают обучать глубокие нейронные сети. Популярные архитектуры моделей временных рядов – **долгая краткосрочная память** (long short-term memory – LSTM), **сверточная нейронная сеть** (CHC) и **трансформер**. Они могут читать на входе временные ряды произвольной длины и генерировать предсказания на основе всей последовательности. Также нейронные сети часто применяются к текстам, которые читаются пословно или посимвольно. Слова и символы обычно представляются **векторами погружений**, которые обучаются на больших корпусах текстовых документов. О погружениях мы будем говорить в разделе 4.7.1.

4.2.7. Проявите свои творческие способности

В начале этого раздела я отмечал, что конструирование признаков – творческий процесс. Аналитик находится в наилучшем положении для определения того, какие признаки считать хорошими в модели предсказания. Поставьте себя на место алгоритма обучения и подумайте, на какие аспекты данных вы стали бы смотреть, решая, какую метку назначить.

Допустим, вы классифицируете сообщения электронной почты как важные и неважные. Можно заметить, что немало важных сообщений поступает от департамента государственных сборов в первый понедельник каждого месяца. Создайте признак «государство первый понедельник». Пусть он будет равен 1, если сообщение поступило от департамента государственных сборов в первый понедельник месяца, и 0 в противном случае. Можно также заметить, что сообщения, содержащие более одного смайлика, редко бывают важными. Создайте признак «содержит смайлики». Пусть он будет равен 1, если сообщение содержит более одного смайлика, и 0 в противном случае.

4.3. ШТАБЕЛИРОВАНИЕ ПРИЗНАКОВ

Вернемся к задаче классификации названий фильмов в твитах. Каждый пример состоит из трех частей:

- 1) пять слов¹, предшествующих выделенному потенциальному названию фильма (левый контекст);
- 2) выделенное потенциальное название фильма (экстракт);
- 3) пять слов, следующих за выделенным потенциальным названием фильма (правый контекст).

Для представления таких многочастных примеров мы сначала преобразуем каждую часть в вектор признаков, а затем штабелируем все три вектора признаков один за другим и получаем вектор признаков для примера в целом.

4.3.1. Штабелирование векторов признаков

В задаче о классификации названий фильмов мы сначала собираем все левые контексты. Затем применяем метод мешка слов, чтобы преобразовать каждый левый контекст в бинарный вектор признаков. Далее собираем все экстракты и с помощью мешка слов преобразуем каждый экстракт в бинарный вектор признаков. После этого то же самое делается с правыми контекстами. Наконец, мы конкатенируем векторы признаков левого контекста, экстракта и правого контекста. Получившийся финальный вектор признаков, представляющий весь пример, показан на рис. 4.13.

Отметим, что все три вектора признаков (по одному из каждой части примера) создаются независимо друг от друга. Это означает, что словари лексем различны для каждой части и потому размерности векторов признаков каждой части могут не совпадать.

Порядок конкатенации векторов признаков не имеет значения. Признаки левого контекста могут быть помещены в середину или в правый конец финального вектора признаков. Однако порядок конкатенации должен быть

¹ На практике контекст слева или справа от потенциального названия фильма в некоторых примерах может быть короче пяти слов, потому что это начало или конец твита.

одинаковым для всех примеров. Это гарантирует, что каждый признак будет представлять одно и то же свойство примеров.

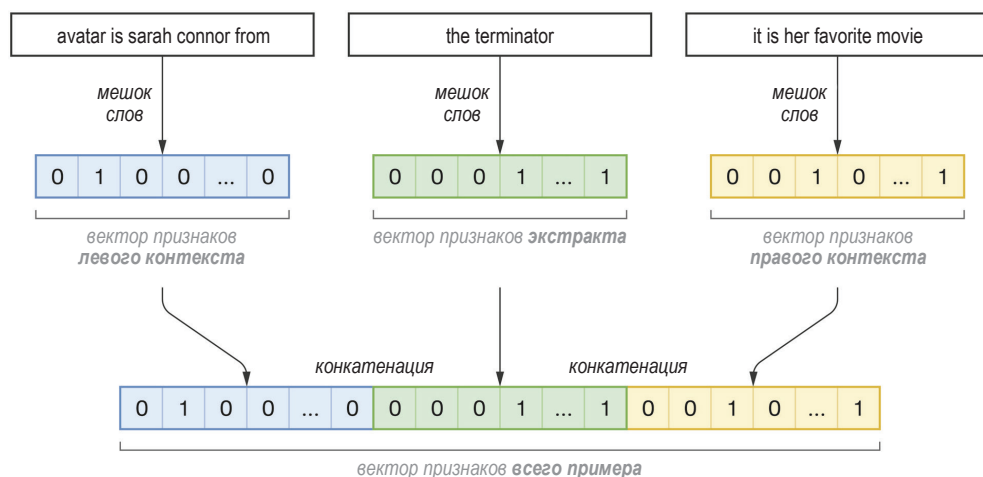


Рис. 4.13 ❖ Создание и штабелирование векторов признаков

4.3.2. Штабелирование индивидуальных признаков

До сих пор мы конструировали признаки скопом. Унитарное кодирование и мешок слов часто порождают тысячи признаков. Это, конечно, весьма эффективный с точки зрения временных затрат способ конструирования признаков, но в некоторых задачах требуется получать векторы признаков с достаточно высокой **предсказательной способностью**. Предсказательную способность признака мы обсудим в следующем разделе.

Допустим, что у нас уже есть классификатор m_A , который принимает весь твит на входе и предсказывает его тему. Пусть одной из тем будет кино. Возможно, мы захотим обогатить векторы признаков в задаче классификации названий фильмов дополнительной информацией, взятой из классификатора m_A . В этом случае мы сконструируем один признак, который можно описать как «является ли темой твита кино», и этот признак будет бинарным: 1, если тема, предсказанная m_A для всего твита, – кино, и 0 в противном случае. И снова конкатенируем все три частичных вектора признаков, как показано на рис. 4.14.

Можно придумать еще много полезных признаков для классификации названий фильмов в твитах, например:

- средний рейтинг фильма в базе данных IMDB;
- количество проголосовавших за фильм в базе данных IMDB;
- рейтинг фильма по версии сайта Rotten Tomato;
- был ли фильм снят недавно (или число, равное году выхода на экран);
- содержит ли текст твита другие названия фильмов;
- содержит ли текст твита имена актеров или режиссеров.

Все эти дополнительные признаки, коль скоро они числовые, можно конкатенировать к вектору признаков. Единственное условие – порядок конкатенации должен быть одинаков для всех примеров.

	Bo-w 1						Bo-w 1												Bo-w M – 1						Bo-w M		Is cinema?
Пример 1	0	1	0	0	...	0	0	0	0	1	...	1	0	0	1	0	...	1	1								
Пример 2	0	0	1	1	...	1	0	1	0	1	...	0	1	1	1	0	...	0	0								
⋮								
Пример N	0	0	1	1	...	0	1	1	0	1	...	1	1	0	1	1	...	1	1								

Рис. 4.14 ❖ Штабелирование одного признака

4.4. СВОЙСТВА ХОРОШИХ ПРИЗНАКОВ

Не все признаки одинаковы. В этом разделе мы поговорим о свойствах хорошего признака.

4.4.1. Высокая предсказательная способность

Прежде всего хороший признак обладает высокой **предсказательной способностью**. В главе 3 мы говорили о предсказательной способности как о свойстве данных. Однако признак тоже может иметь высокую или низкую предсказательную способность. Допустим, требуется предсказать, болен ли пациент раком. Среди прочих признаков мы знаем марку его автомобиля и его семейное положение. Эти два признака вряд ли можно назвать хорошими предикторами рака, поэтому алгоритм машинного обучения не сможет вывести полезную связь между ними и меткой. Предсказательная способность – это свойство признака относительно конкретной задачи. Марка автомобиля и семейное положение могли бы иметь высокую предсказательную способность, но в другой задаче.

4.4.2. Быстрое вычисление

Хорошие признаки можно вычислить быстро. Пусть требуется предсказать тему твита. Твит – короткий текст, поэтому вектор признаков, созданный методом мешка слов, будет разреженным. **Разреженным** называется вектор, большая часть измерений которого содержит нули. Если набор данных мал и тексты короткие, то алгоритму обучения будет трудно найти закономерности в разреженных векторах, потому что они содержат мало информации по сравнению с размером. Информация в одном разреженном векторе редко

находится в тех же позициях, что и в другом разреженном векторе, даже если они представляют похожие концепты.

Чтобы уменьшить разреженность, мы можем пополнить разреженные векторы дополнительными ненулевыми значениями. Для этого можно отправить текст твита в Википедию в качестве поискового запроса, а затем извлечь другие слова из результатов поиска. API Википедии не дает никаких гарантий относительно скорости выполнения запроса, так что на получение ответа может уйти несколько секунд. В системах реального времени выделение признака должно быть быстрым: менее информативный признак, вычисляемый за доли миллисекунды, часто предпочтительнее признака с высокой предсказательной способностью, на вычисление которого уходит несколько секунд. Если приложение должно работать быстро, то признаки, получаемые от Википедии, вряд ли подойдут.

4.4.3. Надежность

Хороший признак должен быть надежным. Как и в примере с Википедией, нет никакой гарантии, что сайт вообще ответит: возможно, он вышел из строя, выключен на плановое обслуживание или временно перегружен и не отвечает на запросы. Поэтому нельзя полагать, что признаки, основанные на Википедии, всегда будут доступны и полны. Так что такие признаки нельзя назвать надежными. Всего один ненадежный признак может снизить качество предсказаний модели. Более того, некоторые предсказания вообще могут стать неправильными, если отсутствует значение важного признака.

4.4.4. Некоррелированность

Два признака **коррелируют**, если их значения как-то связаны. Если рост одного признака влечет за собой рост другого и обратное тоже верно, то эти два признака коррелированы.

После начала эксплуатации модели ее качество может измениться, потому что свойства входных данных со временем меняются. Если многие признаки коррелированы, то даже небольшое изменение характера входных данных может привести к значительным изменениям в поведении модели.

Бывает так, что модель создавалась в условиях нехватки времени, поэтому разработчик использовал все доступные источники признаков. Со временем сопровождение этих источников может стать слишком дорогостоящим занятием. В общем случае рекомендуется исключать избыточные или сильно коррелированные признаки. Уменьшить их количество помогают методы отбора признаков.

4.4.5. Другие свойства

Важное свойство хорошего признака заключается в том, что распределение его значений в обучающем наборе близко к распределению на этапе эксплуа-

тации. Например, для некоторых предсказаний о твите может быть важна дата твита. Однако если применить модель, обученную на старых твитах, для предсказания чего-то относительно текущих твитов, то дата примеров в производственной среде окажется вне интервала дат в обучающих примерах, что может стать причиной серьезных ошибок¹.

Наконец, конструируемые признаки должны быть цельными, простыми для понимания и сопровождения. Цельность означает, что признак представляет величину, которую легко понять и объяснить. Например, при классификации типа автомобиля по его характеристикам можно использовать такие цельные признаки, как вес, длина, ширина и цвет. Признак «длина, поделенная на вес» цельным не является, потому что состоит из двух цельных признаков.

В некоторых алгоритмах обучения комбинирование признаков даже полезно. Однако лучше делать это на специальном этапе конвейера обучения модели. Мы рассмотрим комбинирование признаков и генерирование синтетических признаков ниже в этой главе.

4.5. ОТБОР ПРИЗНАКОВ

Не все признаки будут одинаково важны для решаемой задачи. Например, в задаче определения фильмов в твитах продолжительность фильма вряд ли так уж важна. В то же время при использовании мешка слов словарь может быть очень велик, хотя большинство лексем встречаются в коллекции твитов только один раз. Если алгоритм обучения видит, что некоторый признак отличен от нуля только в паре обучающих примеров, то сомнительно, что он сможет обучиться какой-то полезной закономерности на основе этого признака. Но если вектор признаков очень широкий (содержит тысячи или миллионы признаков), то обучение может недопустимо затянуться. Кроме того, размер обучающих данных окажется настолько большим, что они не поместятся в память стандартного сервера.

Если бы мы могли оценить важность признаков, то сохранили бы только наиболее важные. Это позволило бы сэкономить время, разместить больше примеров в памяти и улучшить качество модели. Ниже описано несколько методов отбора признаков.

4.5.1. Отрезание длинного хвоста

Обычно, если признак содержит информацию (например, имеет ненулевое значение) только для небольшого числа примеров, его можно безболезненно исключить из вектора. В случае **мешка слов** можно построить график, опи-

¹ Информация о датах часто бывает важна для машинного обучения и включается в обучающие данные. Подумайте о **циклических признаках** типа «час суток», «день недели», «месяц года». Для задач предсказания, в которых сезонность имеет предсказательную способность, наличие таких признаков может быть полезно.

сывающий распределение счетчиков лексем, а затем отрезать так называемый длинный хвост, как показано на рис. 4.15.

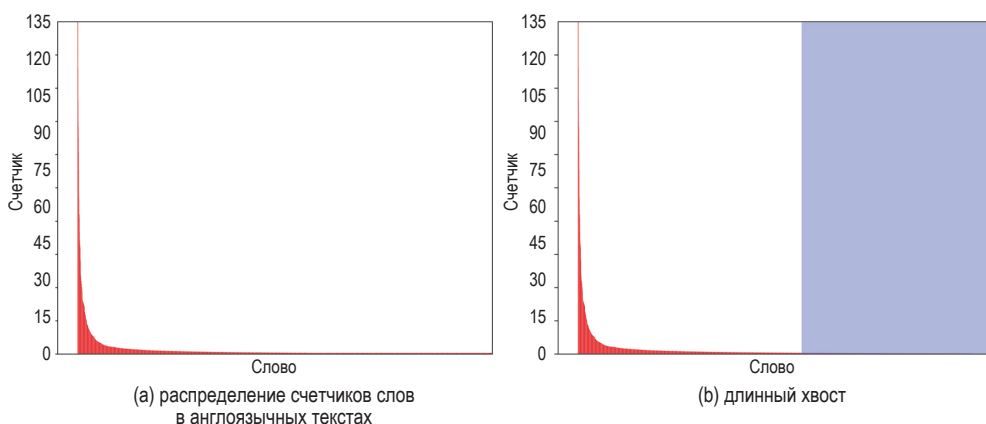


Рис. 4.15 ❖ Распределение счетчиков слов в коллекции англоязычных текстов (а) и длинный хвост (б, голубая зона). Самый большой счетчик соответствует слову «the» (равен 615), самый маленький – слову «zambia» (равен 1)

Длинный хвост распределения – это такая его часть, которая содержит элементы, для которых счетчики значительно меньше, чем для группы элементов с большими счетчиками. Эта меньшая группа называется головой распределения, а сумма счетчиков в ней должна составлять по крайней мере половину суммы всех счетчиков.

Решение о пороге определения длинного хвоста субъективно. Можно сделать его гиперпараметром задачи и найти оптимальное значение экспериментально. С другой стороны, решение можно принять, глядя на распределение счетчиков типа показанного на рис. 4.15(а). Как видите, я отрезал длинный хвост в точке, где распределение элементов становится визуально плоским (рис. 4.15(б)). Отрезать ли длинный хвост и где именно – вопрос дискуссионный. Даже признаки, значения которых редко бывают отличны от нуля, могут оказаться важными. Однако удаление признаков из длинного хвоста может ускорить обучение и породить более качественную модель.

4.5.2. Boruta

Отрезание длинного хвоста – не единственный способ отбора важных признаков и удаления менее важных. В конкурсах **Kaggle** популярен инструмент **Boruta**, который итеративно обучает **случайный лес** и прогоняет **статистические тесты**, чтобы выявить важные и неважные признаки. Этот инструмент существует в виде пакета на R и модуля на Python.

Boruta работает как обертка вокруг алгоритма обучения случайного леса, отсюда и название; Бору́та – это дух леса в славянской мифологии. Чтобы понять, как устроен алгоритм Boruta, вспомним сначала, как работает алгоритм обучения случайного леса.

В основе случайного леса лежит идея **бэггинга**. Он производит много случайных выборок из обучающего набора, а затем обучает статистическую модель на каждой выборке. После этого предсказание делается большинством голосов (в случае классификации) или усреднением (в случае регрессии) по всем моделям. Единственно существенное отличие случайного леса от оригинального алгоритма бэггинга заключается в том, что в первом случае обученными статистическими моделями являются решающие деревья. При каждом разделении решающего дерева рассматривается случайное подмножество всех признаков.

Полезное свойство случайного леса – встроенная способность оценивать важность каждого признака. Ниже я объясню, как этот механизм работает в случае классификации.

Алгоритм состоит из двух этапов. Сначала классифицируются все обучающие примеры из оригинального обучающего набора. Каждое решающее дерево в модели случайного леса голосует только на классификации примеров, которые не использовались при построении этого дерева. После того как дерево протестировано, запоминается, какие из сделанных им предсказаний были правильными.

На втором этапе значения некоторого признака случайным образом переставляются в совокупности примеров и тесты повторяются. Снова для каждого дерева запоминается число правильных предсказаний. Затем вычисляется важность признака для одного дерева как разность количества правильных классификаций при первоначальной конфигурации и после перестановки, поделенная на количество примеров. Для получения общей оценки меры важности признаков для отдельных деревьев усредняются. Удобно, хотя и не обязательно, использовать **z-оценки**, а не первичные оценки важности.

Чтобы найти z-оценку для признака, мы сначала находим среднее значение и стандартное отклонение оценок индивидуального признака для индивидуальных деревьев. Z-оценка признака получается вычитанием среднего значения из первичной оценки и делением результата на стандартное отклонение.

На этом можно остановиться и использовать z-оценки каждого признака в качестве критерия его сохранения (чем выше, тем лучше). Но на практике оценка важности сама по себе часто не отражает существенных корреляций между признаками и целевым показателем. Поэтому нам нужен другой инструмент, который позволил бы отличить действительно важные признаки от неважных, и, как вы уже догадались, Boruta и есть такой инструмент.

Идея Boruta проста: сначала мы расширяем список признаков, добавляя рандомизированную копию каждого оригинального признака, а затем строим классификатор по этому расширенному набору данных. Чтобы оценить важность оригинального признака, мы сравниваем его со всеми рандомизированными признаками. Только те признаки, важность которых выше, чем у рандомизированных – и при этом статистически значимые, – считаются действительно важными.

Ниже перечислены основные шаги алгоритма Boruta, следуя описанию, данному авторами¹, но с небольшими изменениями для большей понятности.

¹ Miron B. Kursa, Aleksander Jankowski, Witold R. Rudnicki. Boruta – A System for Feature Selection // Fundamenta Informaticae 101, 2010, p. 271–285.

Алгоритм Boruta

- Построить расширенные обучающие векторы признаков, в которых каждый оригинальный признак реплицирован. Случайным образом переставить значения реплицированных признаков на обучающих примерах, чтобы исключить корреляцию между реплицированными переменными и целевым показателем.
- Выполнить несколько раундов обучения случайного леса. Реплицированные признаки рандомизируются перед каждым раундом путем применения того же случайного процесса перестановки значений признаков, что и на предыдущем шаге.
- Для каждого раунда вычислить важность (z-оценку) всех оригинальных и реплицированных признаков.
 - Признак считается важным в одном раунде, если его важность больше, чем максимальная важность всех реплицированных признаков.
- Выполнить **статистический тест** для всех оригинальных признаков.
 - **Нулевая гипотеза** заключается в том, что важность признака равна максимальной важности реплицированных признаков (MIRA).
 - В качестве статистического теста используется **двусторонний критерий равенства**: гипотеза может быть отвергнута, если важность признака значительно выше или значительно ниже MIRA.
 - Для каждого оригинального признака подсчитать и запомнить количество успехов.
 - Количество успехов для признака равно числу раундов, в которых важность этого признака оказалась выше MIRA.
 - Математическое ожидание количества успехов для R раундов равно $E(R) = 0.5R$, а стандартное отклонение $S = \sqrt{0.25R}$ (биномиальное распределение с $p = q = 0.5$).
 - Оригинальный признак считается важным (принимается), если число успехов значительно больше ожидаемого, и неважным (отвергается), если оно значительно меньше ожидаемого. (Можно вычислить пределы приемки и отвержения признака для любого числа раундов и желаемого доверительного уровня.)
- Исключить неважные признаки из векторов признаков (как оригинальные, так и реплицированные).
- Выполнить эту процедуру заранее определенное число раз или пока все признаки не будут либо отвергнуты, либо убедительно сочтены важными – в зависимости от того, что произойдет раньше.

Boruta продемонстрировал эффективность во многих конкурсах Kaggle, поэтому считается универсальным инструментом для отбора признаков. Но прежде чем применять его в производственной среде, стоит отметить, что это эвристический алгоритм. Нет никаких теоретических гарантий его качества. Если вы хотите быть уверены, что Boruta не причинит вреда, выполните его несколько раз и убедитесь, что отбор признаков стабилен (т. е. при многократном применении Boruta к вашим данным получаются непротиворечивые результаты). Если отбор признаков нестабилен, то проверьте, достаточно ли велико число деревьев в случайном лесе.

Хотя Boruta – эффективный метод отбора признаков, он не единственный. Описание еще нескольких методов вы найдете в расширенной версии этой главы на сопровождающем книгу вики-сайте.

4.5.3. L1-регуляризация

Регуляризация – общий термин, применяемый к различным методам повышения **обобщаемости** модели. Под обобщаемостью понимается способность модели правильно предсказывать метки ранее не предъявлявшихся примеров.

Хотя регуляризация не позволяет выявлять важные признаки, иные методы, например L1-регуляризация, дают алгоритму обучения возможность игнорировать некоторые признаки.

Порядок применения L1-регуляризации может зависеть от типа обучаемой модели, но основной принцип всегда одинаков: L1 штрафует модель за излишнюю сложность.

На практике L1-регуляризация порождает **разреженную модель**, в которой большинство параметров равны нулю. Поэтому признаки отбираются на основе решения о том, существенны они для предсказания или нет. О регуляризации вообще мы подробнее будем говорить в следующей главе.

4.5.4. Зависящий от задачи отбор признаков

Отбор признаков может зависеть от задачи. Например, из мешка слов, представляющего тексты на естественном языке, можно исключить признаки, соответствующие **стоп-словам**, т. е. таким словам, которые являются слишком общими или слишком часто встречающимися с точки зрения решаемой задачи. Примеры часто встречающихся стоп-слов – артикли, предлоги и местоимения. Словари стоп-слов для большинства языков имеются в сети.

Чтобы еще уменьшить размерность вектора признаков для текстовых данных, иногда подвергают текст предварительной обработке, заменяя редко встречающиеся слова (например, те, для которых счетчик вхождений в корпус текстов меньше трех) одной синтетической лексемой, например RARE_WORD.

4.6. СИНТЕЗИРОВАНИЕ ПРИЗНАКОВ

Алгоритмы обучения, реализованные в самом популярном пакете машинного обучения на Python, **scikit-learn**, работают только с числовыми признаками. Тем не менее бывает полезно преобразовать числовые признаки в категориальные.

4.6.1. Дискретизация признаков

Для дискретизации вещественных признаков может быть много причин. Например, некоторые методы отбора признаков применимы только к категориальным признакам. Успешная дискретизация добавляет полезную для алгоритма обучения информацию в случаях, когда обучающий набор относительно мал. Многочисленные исследования показывают, что дискретизация может повышать верность предсказаний. Кроме того, человеку проще интерпретировать предсказание модели, если оно основано на дискретных группах значений, например возрастных группах или диапазонах заработной платы.

Распределение по интервалам (binning), или **группировка**, – популярный метод преобразования числового признака в категориальный путем замены числовых значений, попадающих в определенный диапазон, постоянным категориальным значением.

Есть три подхода к группировке:

- равномерная группировка;
- группировка методом k средних;
- квантильная группировка.

Во всех трех случаях необходимо принять решение о числе интервалов. Рассмотрим иллюстрацию на рис. 4.16. Здесь имеется числовой признак j и 12 его значений, по одному для каждого из 12 примеров в наборе данных. Допустим, мы решили завести три интервала. При равномерной группировке все интервалы будут иметь одинаковую ширину, как на верхнем рис. 4.16.

При группировке методом k средних значения в каждом интервале принадлежат ближайшему одномерному кластеру, построенному методом k средних, как на среднем рис. 4.16.

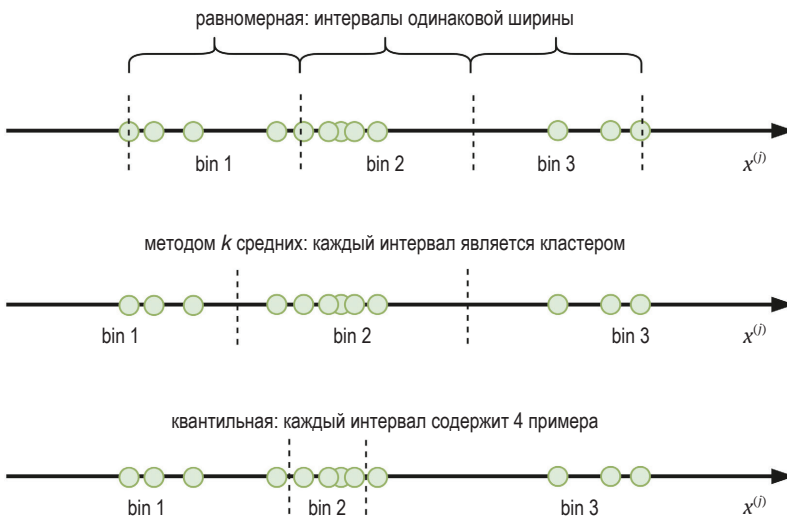


Рис. 4.16 ❖ Три способа группировки: равномерная, методом k средних и квантильная

При квантильной группировке во всех интервалах одинаковое количество примеров, как на нижнем рис. 4.16.

В случае равномерной группировки, если после развертывания модели в производственной среде оказывается, что значение признака во входном векторе меньше самого левого или больше самого правого интервала, то выбирается ближайший интервал, т. е. соответственно самый левый или самый правый.

Напомним, что в большинстве реализаций современных алгоритмов машинного обучения признаки должны быть числовыми. Поэтому интервалы необходимо преобразовать обратно к числовым значениям, воспользовавшись, например, **унитарным кодированием**.

4.6.2. Синтез признаков из реляционных данных

Аналитики часто работают с данными из **реляционных баз данных**. Например, мобильному оператору нужно знать, откажется ли абонент от подписки в ближайшем будущем. Эта задача называется **анализом оттока клиентов**. Мы должны представить каждого абонента вектором признаков.

Предположим, что данные о пользователях собраны в трех реляционных таблицах: User, Order и Call, – как показано на рис. 4.17.

User				
User ID	Gender	Age	...	Date Subscribed
1	M	18	...	2016-01-12
2	F	25	...	2017-08-23
3	F	28	...	2019-12-19
4	M	19	...	2019-12-18
5	F	21	...	2016-11-30

Order				
Order ID	User ID	Amount	...	Order Date
1	2	23	...	2017-09-13
2	4	18	...	2018-11-23
3	2	7.5	...	2019-12-19
4	2	8.3	...	2016-11-30

Call				
Call ID	User ID	Call Duration	...	Call Date
1	4	55	...	2016-01-12
2	2	235	...	2016-01-13
3	3	476	...	2016-12-17
4	4	334	...	2019-12-19
5	4	14	...	2016-11-30

Рис. 4.17 ❖ Реляционные данные для анализа оттока клиентов

Таблица User содержит два потенциально полезных признака: Gender (пол) и Age (возраст). Синтетические признаки можно создать также по данным в таблицах Order и Call. Как видим, для пользователя 2 в таблице Order имеются две строки, а для пользователя 4 – одна строка в таблице Order, зато три строки в таблице Calls. Чтобы создать признак, представляющий одного пользователя, мы должны свести эти несколько записей в одно значение. Типичный подход – вычислить различные статистики данных из нескольких строк и использовать значение каждой статистики в качестве признака. Чаще всего используются **выборочное среднее** и **стандартное отклонение**. (Стандартное отклонение равно квадратному корню из **выборочной дисперсии**.)

В качестве конкретного примера я вычислил значения четырех признаков для пользователей 2 и 4. Они показаны на рис. 4.18.

User features

User ID	Gender	Age	Mean Order Amount	Std Dev Order Amount	Mean Call Duration	Std Dev Call Duration
2	F	25	12.9	7.1	235	0
4	M	19	18	0	134.3	142.7

Рис. 4.18 ❖ Синтетические признаки на основе выборочного среднего и стандартного отклонения

Иногда реляционная база данных имеет более глубокую структуру. Например, у пользователя может быть несколько заказов, а в каждом заказе может быть несколько позиций. В таком случае можно вычислить статистику статистики. Например, создать признак, вычислив стандартное отклонение цен позиций в каждом заказе, а затем усреднить эти стандартные отклонения для конкретного пользователя. Статистики можно комбинировать как угодно: среднее среднего, стандартное отклонение среднего, стандартное отклонение стандартного отклонения и т. д. Тот же принцип применим к базе данных, в которой имеется более двух уровней таблиц.

Сгенерировав признаки на основе всех возможных комбинаций статистик, мы можем отобрать наиболее полезные, воспользовавшись каким-нибудь методом отбора признаков.

Если вы хотите повысить предсказательную способность векторов признаков или если обучающий набор относительно мал, то можно синтезировать дополнительные признаки, полезные для предсказаний. Существует два основных метода синтеза: по данным или по другим признакам.

4.6.3. Синтезирование признаков по данным

Для синтеза одного или нескольких дополнительных признаков часто применяется **кластеризация**. Воспользуемся кластеризацией методом **k средних**. Выберите значение k . Если конечная цель – построить модель классификации, обычно в качестве k берут число классов C . В случае регрес-

сии обратитесь к своей интуиции или примените какой-нибудь метод для определения правильного числа кластеров, например **предсказательную силу** или **метод локтя**. Примените кластеризацию методом k средних к векторам признаков в обучающих данных. Затем добавьте k дополнительных признаков к уже имеющимся векторам признаков. Дополнительный признак $D + j$, где $j = 1, \dots, k$, будет бинарным и равным 1, если соответствующий вектор признаков принадлежит кластеру j .

Можно синтезировать еще больше признаков, применив другие алгоритмы кластеризации или запустив алгоритм k средних несколько раз с разными случайно выбранными начальными точками.

4.6.4. Синтезирование признаков по другим признакам

Нейронные сети известны своей способностью обучаться сложным признакам путем комбинирования простых признаков нетривиальными способами. Простые признаки проходят несколько уровней нелинейных преобразований. Если данных в изобилии, то можно обучить глубокий **многослойный перцептрон** хитроумно комбинировать базовые цельные признаки, полученные на входе.

Если у вас нет бесконечного запаса обучающих примеров (а на практике так часто и бывает), то очень глубокие нейронные сети теряют свою привлекательность¹. Если набор данных не очень большой (от тысячи до сотни тысяч примеров), то, возможно, стоит предпочесть поверхностный алгоритм обучения и «помочь» ему, предоставив обогащенное множество признаков.

На практике для получения новых признаков из существующих чаще всего применяют какое-нибудь простое преобразование к одному или паре признаков. Есть три таких типичных преобразования, применяемых к числовому признаку j примера i : 1) **дискретизация** признака; 2) возведение признака в квадрат и 3) вычисление выборочного среднего и стандартного отклонения признака j от k ближайших соседей примера i в смысле некоторой метрики, например евклидова расстояния или коэффициента Отиаи.

К паре числовых признаков применяются следующие простые операции: $+$, $-$, \times и \div (эта техника называется также **скрещиванием признаков**). Например, значение нового признака q в примере i , где $q > D$, можно получить, скомбинировав значения признаков 2 и 6 следующим образом: $x_i^{(q)} \stackrel{\text{def}}{=} x_i^{(2)} \div x_i^{(6)}$. Я выбрал признаки 2 и 6 и оператор \div совершенно произвольно. Если число D оригинальных признаков не слишком велико, то можно сгенерировать все возможные преобразования (рассмотрев все пары признаков и все арифметические операторы). Затем, воспользовавшись каким-то методом отбора признаков, выберите те, что повышают качество модели.

¹ Если только вы не используете глубокие предобученные модели для переноса обучения, которое мы обсудим в следующей главе.

4.7. ОБУЧЕНИЕ ПРИЗНАКОВ НА ДАННЫХ

Иногда полезные признаки можно обучить на данных. Это особенно эффективно, когда имеется доступ к большим коллекциям релевантных помеченных или непомеченных данных, например корпуса текстов или наборы изображений, взятых из веба.

4.7.1. Погружения слов

В главе 3 мы использовали погружения слов для приращения данных. **Погружениями слов** называются векторы признаков, представляющие слова. Для похожих слов соответствующие векторы признаков похожи, причем сходство определяется с помощью некоторой меры, например **коэффициента Отиаи**. Погружения слов обучаются на большом корпусе текстовых документов. **Поверхностная нейронная сеть** с одним скрытым слоем (так называемый **слой погружения**) обучается предсказывать слово по окружающим словам или предсказывать окружающие слова по среднему слову. После обучения нейронной сети параметры слоя погружения используются в качестве погружений слов. Существует много алгоритмов обучения погружений слов на данных. Самый известный, предложенный Google, называется **word2vec**, его исходный код опубликован. Предобученные этим алгоритмом погружения для многих языков доступны для скачивания.

Имея коллекцию погружений слов для некоторого языка, можно использовать ее для представления отдельных слов в предложениях и документах, написанных на этом языке, вместо **унитарного кодирования**.

Посмотрим, как обучаются погружения слов одним из вариантов алгоритма word2vec, называемым **скипграммами**. Наша цель – построить модель, которую можно будет использовать для преобразования унитарного кода слова в погружение. Допустим, что словарь содержит 10 000 слов. Унитарный вектор для каждого слова – это вектор длиной 10 000, во всех позициях которого, кроме одной, находятся нули, а в этой последней 1. Разным словам соответствуют векторы, содержащие 1 в разных позициях.

Рассмотрим предложение «I am attentively reading the book on machine learning engineering»¹. Возьмем то же самое предложение, но удалим одно слово, скажем «book». Предложение принимает вид «I am attentively reading the · on machine learning engineering». Теперь оставим только три слова перед · и три слова после : «attentively reading the · on machine learning». Если я попрошу вас, глядя на это шестисловное окно, угадать, что нужно подставить вместо ·, вы, наверное, скажете: «book», «article» или «paper». Именно так контекстные слова позволяют предсказать слово, которое они окружают. И именно так машина может обучиться тому, что слова «book», «paper» и «article» схожи по смыслу. Они встречаются в похожих контекстах во многих текстах.

¹ Я внимательно читаю книгу по инженерии машинного обучения. – Прим. перев.

Оказывается, что верно и обратное: по слову можно предсказать окружающий его контекст. Фрагмент «attentively reading the · on machine learning» называется скипграммой с окном размера 6 (3 + 3). Пользуясь документами, доступными в вебе, мы легко можем создать сотни миллионов скипграмм.

Будем обозначать скипграмму следующим образом: $[x_{-3}, x_{-2}, x_{-1}, x, x_{+1}, x_{+2}, x_{+3}]$. В нашем предложении x_{-3} – унитарный вектор для слова «attentively», x_{-2} – для слова «reading», x – для пропущенного слова (·), x_{+1} – для слова «on» и т. д.

Скипграмма с окном размера 4 будет выглядеть так: $[x_{-2}, x_{-1}, x, x_{+1}, x_{+2}]$. Ее можно схематически изобразить, как показано на рис. 4.19. Это **полносвязная сеть**, как и **многослойный перцептрон**. Входное слово обозначается в скипграмме символом ·. Нейронная сеть должна обучиться предсказывать контекстные слова по центральному слову.

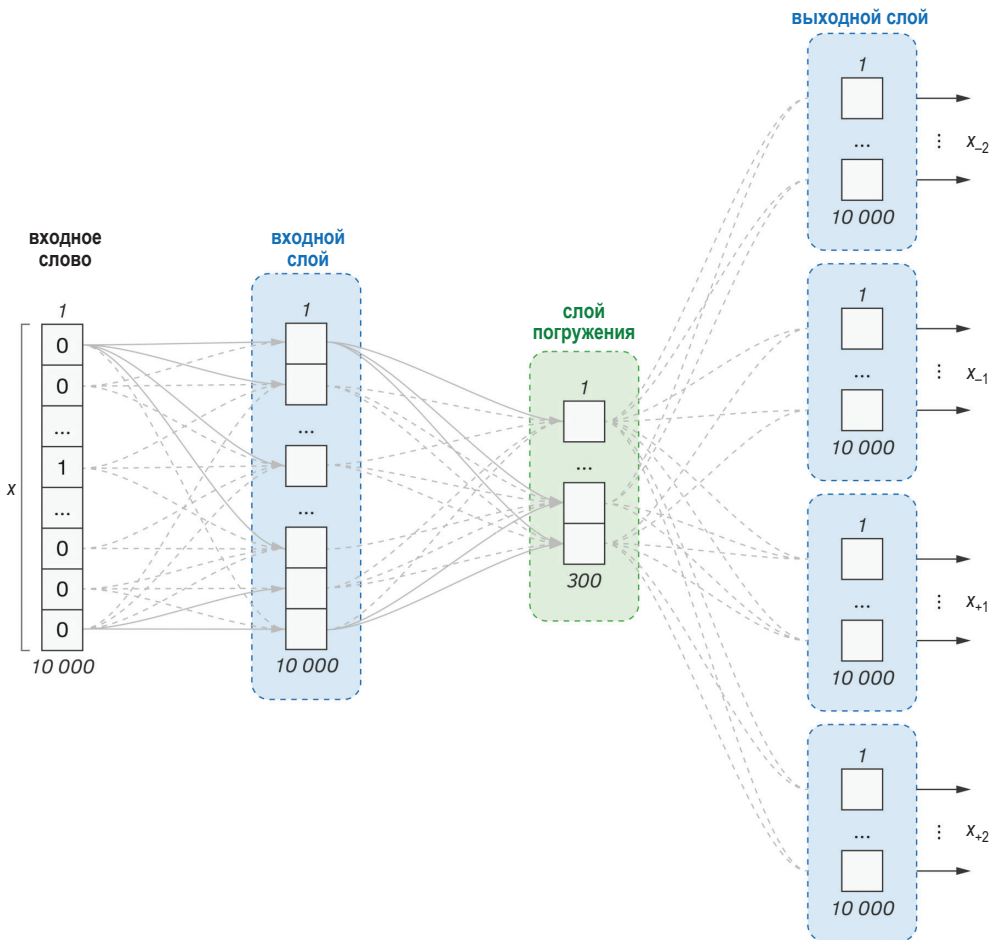


Рис. 4.19 ❖ Скипграммная модель с размером окна 4 и слой погружения с 300 блоками

В качестве **функции активации** в выходном слое используется **softmax**, а в качестве **функции стоимости** – **отрицательное логарифмическое правдоподобие**. Погружение слова определяется параметрами слоя погружения, применяемыми, когда унитарный код этого слова подается на вход модели.

У погружений слов, обученных алгоритмом word2vec, есть проблема – множество погружений фиксировано, а модель нельзя использовать для внесловарных слов, т. е. слов, которых не было в корпусе, на котором обучались погружения. Существуют другие архитектуры нейронных сетей, позволяющие получить погружение для любого слова, даже внесловарного. Одна такая архитектура, часто применяемая на практике, называется **fastText**. Она была придумана в компании Facebook, ее исходный код открыт.

Ключевое различие между word2vec и fastText заключается в том, что в word2vec каждое слово в корпусе рассматривается как цельная сущность и для каждого слова обучается вектор. А в fastText каждое слово трактуется как среднее векторов погружений, представляющих символьные n -граммы, составляющие это слово. Например, погружением для слова «mouse» является среднее векторов погружения n -грамм «<mo>», «mou», «<mou>», «mous», «<mous>», «mouse», «<mouse>», «mouse>», «ous», «ouse», «ouse>», «use», «use>», «se>» (в предположении, что длины наименьшей и наибольшей n -грамм равны соответственно 3 и 6).

Погружения слов – эффективный способ представления текстов на естественном языке для использования в таких архитектурах нейронных сетей, как **рекуррентные нейронные сети** (РНС) и **сверточные нейронные сети** (СНС), адаптированных для работы с последовательностями. Однако если мы хотим использовать погружения слов для представления текстов переменной длины в алгоритмах **поверхностного обучения** (которым требуются входные векторы признаков фиксированной размерности), то придется применить какую-то операцию агрегирования к векторам слов, например вычисление взвешенной суммы или среднего. Представление текстового документа, полученное усреднением слов, составляющих этот документ, на практике оказывается не очень полезным.

4.7.2. Погружения документов

Популярный способ получить погружение предложения или целого документа – воспользоваться архитектурой нейронной сети **doc2vec**, также придуманной в Google и доступной в виде исходного кода. Архитектура doc2vec очень похожа на word2vec. Единственное существенное отличие – тот факт, что теперь имеется два вектора погружения – один для идентификатора документа, другой для слова. Предсказание окружающих слов для входного слова делается в два этапа: сначала оба вектора погружения (документа и слова) усредняются, а затем по этому среднему предсказываются окружающие слова. Чтобы два вектора можно было усреднить, они должны иметь одинаковую размерность. Интересно, что по этой причине можно сравнивать не только векторы документов (вычисляя коэффициент Отиаи), но также векторы документа и слова. Векторы слов, обученные таким способом, очень похожи на векторы, обученные методом word2vec.

Чтобы получить погружения для нового документа, не принадлежащего корпусу, на котором обучались погружения слов, этот новый документ сначала добавляется в корпус. Ему присваивается новый идентификатор документа. Затем существующая модель дообучается на протяжении нескольких эпох, причем все ранее обученные параметры замораживаются, а новые, соответствующие новому идентификатору, обучаются. Идентификатор входного документа предоставляется в виде унитарного кода.

4.7.3. Погружения всего, чего угодно

Следующая техника обычно используется для получения векторов погружения для любого объекта (а не только слов или документов). Сначала мы ставим задачу обучения с учителем, которая принимает на входе объекты и выводит предсказание. Затем строим помеченный набор данных и обучаем нейросетевую модель, которая решает нашу задачу. После этого выходы одного из полносвязных слоев рядом с выходным слоем сети (перед нелинейностью) используются как погружения входного объекта.

Например, помеченный набор изображений ImageNet и глубокая сверточная сеть с архитектурой типа **AlexNet** часто используются для обучения погружений изображений. На рис. 4.20 показаны слои погружения для изображений. Здесь мы имеем глубокую СНС с двумя **полносвязными слоями** перед выходным слоем. Нейронная сеть была обучена предсказывать изображенный объект. Чтобы получить погружение изображения, не предъявлявшегося на этапе обучения модели, мы передаем изображение (обычно представленное тремя матрицами пикселей, по одной на каждый канал R, G, B) на вход нейронной сети, а затем используем выход одного из полносвязных слоев перед нелинейностью. Какой из полносвязных слоев лучше, зависит от поставленной задачи, этот вопрос должен решаться экспериментально.

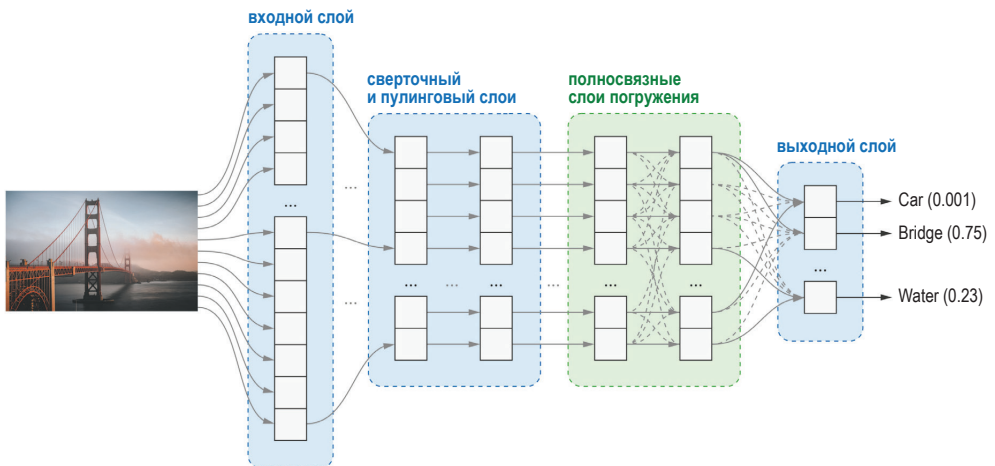


Рис. 4.20 ❖ Архитектура нейронной сети для обучения погружения изображений. Слои погружения показаны зеленым цветом

Следуя описанному выше подходу, мы можем обучить погружения любого типа. Аналитик должен только ответить на три вопроса:

- какую задачу обучения с учителем следует решить (в случае изображений это обычно классификация изображенного объекта);
- как представлять вход нейронной сети (для изображений – матрицами пикселей, по одной на каждый канал);
- какой должна быть архитектура нейронной сети перед полносвязными слоями (для изображений – обычно СНС).

4.7.4. Выбор размерности погружения

Размерность погружения обычно определяется экспериментально или на основе предшествующего опыта. Например, Google в документации по TensorFlow рекомендует следующее эвристическое правило:

$$d = \sqrt[4]{D},$$

где d – размерность погружения, а D – «число категорий». Число категорий для погружений слов равно числу уникальных слов в корпусе. Для произвольных погружений это размерность оригинального входа. Например, если число уникальных слов в корпусе $D = 5\,000\,000$, то размерность погружения $d = \sqrt[4]{5\,000\,000} = 47$. На практике часто используются значения от 50 до 600.

Теоретически более обоснованный подход к выбору размерности погружения – рассматривать ее как гиперпараметр, настраиваемый в последующей задаче. Например, если имеется размеченный корпус документов, то размерность погружения можно оптимизировать посредством минимизации числа ошибок предсказания, допущенных классификатором, обученным на помеченных данных, в которых слова документов представлены погружениями.

4.8. Понижение размерности

Иногда бывает необходимо понизить размерность примеров. Эта задача отличается от отбора признаков. В последнем случае мы анализируем свойства всех существующих признаков и исключаем те, что, по нашему мнению, не вносят существенный вклад в качество модели. А применяя методы **понижения размерности** к набору данных, мы заменяем весь оригинальный вектор признаков новым вектором меньшей размерности, составленным из синтетических признаков.

Понижение размерности часто приводит к ускорению обучения и лучшей обобщаемости.

Кроме того, набор данных становится удобнее для восприятия, ведь человек видит только три измерения. Существует несколько способов понизить размерность. Какой выбрать, зависит от того, что мы хотим сделать. Методы понижения размерности хорошо описаны в теоретических книгах по машинному обучению, поэтому я расскажу только о том, когда аналитик должен предпочесть тот или иной метод.

4.8.1. Быстрое понижение размерности методом PCA

Метод главных компонент (PCA) – самый старый из существующих. И кстати, самый быстрый. Сравнительные тесты показывают очень слабую зависимость быстродействия PCA от размера набора данных. Следовательно, мы можем использовать PCA на шаге, предшествующем обучению модели, и найти оптимальное значение пониженной размерности экспериментально как часть процесса настройки гиперпараметров.

Основной недостаток PCA в том, что данные должны целиком помещаться в память. Существует и вариант PCA, свободный от этого ограничения; он называется **инкрементным PCA** и позволяет применять алгоритм к пакетам набора данных, загружая по одному пакету за раз. Но инкрементный PCA на порядок медленнее PCA. Кроме того, PCA не так полезен для целей визуализации, как два других метода, рассматриваемых ниже.

4.8.2. Понижение размерности с целью визуализации

Если нашей целью является визуализация, то лучше предпочесть алгоритм равномерной аппроксимации и проецирования многообразия (Uniform Manifold Approximation and Projection – **UMAP**), или **автокодировщик**. Тот и другой можно запрограммировать для порождения двумерных или трехмерных векторов признаков, тогда как PCA порождает D так называемых **главных компонент** (где D – размерность данных), и аналитик должен выбрать первые две или три главные компоненты в качестве визуализируемых признаков. В общем случае UMAP работает гораздо быстрее автокодировщика, но получающиеся визуализации выглядят совершенно по-разному, так что выбор одного из двух этих алгоритмов должен основываться на свойствах конкретного набора данных. Кроме того, UMAP, как и PCA, требует, чтобы все данные находились в памяти, тогда как автокодировщика можно обучать на пакетах.

Понижение размерности может также зависеть от задачи. Например, понижать размерность фотографий можно с помощью графического редактора. Аналогично можно уменьшить скорость передачи данных и число каналов в звуковых последовательностях.

4.9. МАСШТАБИРОВАНИЕ ПРИЗНАКОВ

Преобразовав все признаки в числовую форму, мы почти готовы приступить к работе над моделью. Но есть еще один шаг, который может оказаться полезным, – масштабирование признаков.

Под **масштабированием признаков** понимается приведение всех признаков к одинаковым или очень похожим диапазонам значений либо рас-

пределениям. Многочисленные эксперименты показали, что применение алгоритма обучения к масштабированным признакам может порождать более качественную модель. Хотя не гарантируется, что масштабирование даст положительный эффект, оно все же рекомендуется. Кроме того, масштабирование может ускорить обучение глубоких нейронных сетей. Оно также гарантирует, что никакой отдельный признак не будет доминировать, особенно на начальных итерациях градиентного спуска или другого итеративного алгоритма оптимизации. Наконец, масштабирование уменьшает риск **численного переполнения**, возникающего, когда компьютер работает с очень большими или очень малыми числами.

4.9.1. Нормировка

Нормировкой называется процесс преобразования фактического диапазона значений, принимаемых признаком, в предопределенный искусственный диапазон, обычно интервал $[-1, 1]$ или $[0, 1]$.

Например, пусть естественный диапазон признака – от 350 до 1450. Если вычесть 350 из каждого значения признака и поделить разность на 1100, то мы приведем значения к диапазону $[0, 1]$. В общем случае формула нормировки имеет вид:

$$\bar{x}^{(j)} \leftarrow \frac{x^{(j)} - \min^{(j)}}{\max^{(j)} - \min^{(j)}},$$

где $x^{(j)}$ – исходное значение признака j в некотором примере, а $\min^{(j)}$ и $\max^{(j)}$ – соответственно минимальное и максимальное значения признака j в обучающих данных.

Если вы предпочитаете диапазон $[-1, 1]$, то формула нормировки будет иметь вид:

$$\bar{x}^{(j)} \leftarrow \frac{2 \times x^{(j)} - \max^{(j)} - \min^{(j)}}{\max^{(j)} - \min^{(j)}}.$$

Недостаток нормировки в том, что значения $\max^{(j)}$ и $\min^{(j)}$ обычно являются выбросами, поэтому нормировка «сжимает» исходные значения признаков в очень маленький диапазон. Одно из решений этой проблемы – применить **отсечение**, т. е. выбрать «разумные» значения $\max^{(j)}$ и $\min^{(j)}$ вместо экстремальных, встречающихся в обучающих данных. Допустим, мы оценили, что разумный диапазон значений признака – $[a, b]$. Прежде чем вычислять масштабированное значение по одной из приведенных выше формул, значение $x^{(j)}$ признака усекается, т. е. полагается равным a , если $x^{(j)}$ меньше a , и b , если оно больше b . Для оценки значений a и b часто применяется **винсоризация**. Этот метод получил название в честь инженера и биостатистика Чарльза Винсора (1895–1951). Идея заключается в том, чтобы усесть все выбросы, выходящие за пределы определенного процентиля; например, при 90%-ной винсоризации все значения, меньшие 5-го процентиля, будут установлены

равными 5-му процентилю, а все данные, большие 95-го процентиля, будут установлены равными 95-му процентилю. В коде на Python винсоризацию можно применить к списку чисел следующим образом:

```
1 from scipy.stats.mstats import winsorize
2 winsorize(list_of_numbers, limits=[0.05, 0.95])
```

На выходе функции `winsorize` будет список той же длины, что на входе, но значения выбросов будут усечены. Соответствующий код на R выглядит так:

```
1 library(DescTools)
2 DescTools::Winsorize(vector_of_numbers, probs = c(0.05, 0.95))
```

Иногда используется **нормировка по среднему значению**:

$$\bar{x}^{(j)} \leftarrow \frac{x^{(j)} - \mu^{(j)}}{\max^{(j)} - \min^{(j)}},$$

где $\mu^{(j)}$ – выборочное среднее значение признака j .

4.9.2. Стандартизация

Стандартизация (или **нормировка по z-оценке**) – процедура масштабирования значений признаков, в результате которой они приобретают свойства **стандартного нормального распределения** с $\mu = 0$ и $\sigma = 1$, где μ – **выборочное среднее** (значение признака, усредненное по всем обучающим примерам), а σ – стандартное отклонение от выборочного среднего.

Стандартные оценки (или **z-оценки**) признаков вычисляются следующим образом:

$$\hat{x}^{(j)} \leftarrow \frac{x^{(j)} - \mu^{(j)}}{\sigma^{(j)}},$$

где $\mu^{(j)}$ – выборочное среднее значений признака j , а $\sigma^{(j)}$ – **стандартное отклонение** значений признака j от выборочного среднего.

Кроме того, иногда полезно применить простые математические преобразования к значениям признаков еще до описанного выше масштабирования. К таким преобразованиям относятся взятие логарифма, возведение в квадрат или извлечение квадратного корня. Идея в том, чтобы получить распределение, максимально близкое к нормальному.

Возникает вопрос: когда использовать нормировку, а когда стандартизацию. На него нет однозначного ответа. Теоретически нормировка должна работать лучше для равномерно распределенных данных, а стандартизация – для нормально распределенных. Но на практике данные редко имеют идеальное распределение. Если набор данных не слишком велик и у вас есть время, можно попробовать оба подхода и посмотреть, какой работает лучше на ваших данных. Масштабирование признаков обычно улучшает результаты большинства алгоритмов обучения.

4.10. ПРОСАЧИВАНИЕ ДАННЫХ ПРИ КОНСТРУИРОВАНИИ ПРИЗНАКОВ

При конструировании признаков просачивание данных может иметь место в нескольких ситуациях, включая дискретизацию и масштабирование.

4.10.1. Возможные проблемы

Допустим, что мы используем весь набор данных для вычисления ширины каждого интервала или масштабных коэффициентов для признаков. Затем набор данных разбивается на обучающий, контрольный и тестовый. Если так и поступить, то значения признаков в обучающих данных будут отчасти получены с использованием примеров, принадлежащих зарезервированному набору. Если набор данных достаточно мал, то итогом может стать чрезмерно оптимистическая оценка качества модели на зарезервированных данных.

Теперь допустим, что мы работаем с текстом и используем **мешок слов** для создания признаков на основе всего набора данных. После построения словаря данные разбиваются на три набора. В этой ситуации алгоритму обучения будут предъявлены признаки, основанные на лексемах, присутствующих только в зарезервированных наборах. И снова модель продемонстрирует искусственно завышенное качество – лучше, чем если бы данные разбивались на части до конструирования признаков.

4.10.2. Решение

Решение, как вы, наверное, уже поняли, состоит в том, чтобы сначала разбить весь набор данных на обучающий и зарезервированные и конструировать признаки только на основе обучающих данных. Это применимо и к случаю, когда мы используем **кодирование средним**, чтобы преобразовать категориальный признак в число: сначала разделите данные на части, а затем вычисляйте выборочное среднее метки, используя только обучающие данные.

4.11. ХРАНЕНИЕ И ДОКУМЕНТИРОВАНИЕ ПРИЗНАКОВ

Даже если вы планируете обучать модель сразу после конструирования признаков, рекомендуется разработать **файл схемы**, в котором будут описаны ожидаемые свойства признаков.

4.11.1. Файл схемы

Файл схемы – это документ с описанием признаков. Он машиночитаемый, версионный и обновляется после каждого существенного изменения признаков. Вот несколько примеров свойств, которые можно закодировать в схеме:

- имена признаков;
- для каждого признака:
 - тип (категориальный, числовой);
 - доля примеров, в которых этот признак предположительно будет присутствовать;
 - минимальное и максимальное значения;
 - выборочное среднее и дисперсия;
 - допустимы ли нулевые значения;
 - допустимы ли неопределенные значения.

Ниже приведен файл схемы для четырехмерного набора данных:

```

1 feature {
2   name : "height"
3   type : float
4   min : 50.0
5   max : 300.0
6   mean : 160.0
7   variance : 17.0
8   zeroes : false
9   undefined : false
10  popularity : 1.0
11 }
12
13 feature {
14   name : "color_red"
15   type : binary
16   zeroes : true
17   undefined : false
18   popularity : 0.76
19 }
20
21 feature {
22   name : "color_green"
23   type : binary
24   zeroes : true
25   undefined : false
26   popularity : 0.65
27 }
28
29 feature {
30   name : "color_blue"
31   type : binary
32   zeroes : true
33   undefined : false
34   popularity : 0.81
35 }

```

4.11.2. Хранилище признаков

В крупных распределенных организациях может существовать **хранилище признаков**, предназначенное для хранения, документирования, повторного использования и общего доступа со стороны нескольких научных коллективов и проектов. Способы сопровождения и использования признаков в режиме эксплуатации могут существенно зависеть от проекта и коллектива. Это увеличивает сложность инфраструктуры и часто приводит к дублированию работы. Крупные распределенные организации сталкиваются со следующими проблемами.

Признаки не используются повторно

Признаки, представляющие один и тот же атрибут некоторой сущности, реализуются несколько раз разными инженерами и группами, хотя уже проделанную работу и созданные конвейеры машинного обучения можно было бы использовать повторно.

Определения признаков варьируют

Разные группы определяют признаки по-разному, и не всегда есть возможность получить доступ к документации признака.

Признаки требуют большого объема вычислений

Некоторые модели машинного обучения основаны на информативных, но при этом вычислительно дорогих признаках. Размещение таких признаков в быстром хранилище позволило бы использовать их в реальном времени, а не только в пакетном режиме.

Несогласованность между обучением и эксплуатацией

Модель обычно обучается на исторических данных, но на этапе эксплуатации ей предъявляются актуальные данные в режиме реального времени. Значения некоторых признаков могут зависеть от всего исторического набора данных, который во время эксплуатации недоступен. Чтобы модель работала правильно, каждый признак должен иметь одинаковое значение в одинаковых входных примерах как в офлайн-режиме (на этапе разработки), так и в онлайн-режиме (на этапе эксплуатации).

Неизвестен срок хранения признака

Когда в производственной среде поступает новый входной пример, мы не можем точно сказать, какие признаки необходимо пересчитать; приходится прогонять весь конвейер, чтобы вычислить значения всех признаков, необходимых для предсказания.

Хранилище признаков – центральное место для хранения документированных, тщательно отобранных признаков с контролем доступа. Вместе с каждым признаком хранятся четыре элемента: 1) имя, 2) описание, 3) метаданные, 4) определение.

Имя признака – строка, однозначно идентифицирующая признак, например «average_session_length» или «document_length».

Описание признака составляется на естественном языке и описывает свойство, представляемое признаком, например: «Средняя продолжительность сеанса пользователя» или «количество слов в документе».

Помимо этих атрибутов, в файле схемы могут присутствовать метаданные, сообщающие следующую информацию: почему признак был включен в модель, как он способствует обобщаемости, лицо в организации, отвечающее за сопровождение источника данных о признаке¹, тип входных данных (например, числовые, строковые, изображение), тип выходных данных (например, числовой скаляр, категориальные, вектор чисел), должно ли в хранилище кешироваться значение признака, и если да, в течение какого срока. Признак также можно пометить как доступный в онлайнном и офлайнном режимах или только для офлайнной обработки. Признаки, доступные в онлайнном режиме, должны быть реализованы таким образом, что их значение либо 1) быстро читается из кеша или хранилища значений, либо 2) вычисляется в реальном времени. К числу признаков, вычисляемых в реальном времени, относятся, например, возведение входного числа в квадрат, определение формы слова или поиск во внутренней сети организации.

Определение признака представляет собой версионированный код на таком языке, как Python или Java. Он будет выполняться в режиме эксплуатации и применяться к входным данным для вычисления значения признака.

Хранилище признаков позволяет инженерам по данным вставлять новые признаки. В свою очередь, аналитики данных и инженеры по машинному обучению используют API для получения релевантных, на их взгляд, признаков. Хранилище может предоставлять признаки для одиночного онлайнного входа. Или же аналитик, работающий с моделью в офлайнном режиме, может преобразовать обучающие данные в коллекцию векторов признаков и отправить в хранилище пакет входов.

С целью **воспроизводимости** для значений признаков в хранилище ведется контроль версий. Это позволяет аналитику данных заново построить модель с теми значениями признаков, которые использовались при обучении предыдущей версии модели. После обновления значения признака для данного входа предыдущее значение не стирается, а сохраняется вместе с временной меткой, показывающей, когда это значение было сгенерировано. Кроме того, признак j , использованный в модели m_B , сам может являться выходом некоторой модели m_A . При изменении модели m_A важно сохранять ее старые версии: модель m_B может по-прежнему ожидать, что ей на вход будут поданы выходы, сгенерированные прежней версией m_A .

Хранилище признаков является частью конвейера машинного обучения, как показано на рис. 4.21. Эта архитектура навеяна платформой машинного обучения Michelangelo, разработанной компанией Uber. Она содержит два хранилища признаков, онлайнное и офлайнное, данные в которых синхронизируются. В Uber онлайнное хранилище часто обновляется, почти в режиме реального времени, для чего используются данные реального времени. Напротив, офлайнное хранилище обновляется в пакетном режиме

¹ Если ответственный за признаки покидает компанию, владельцу продукта автоматически должно быть направлено уведомление.

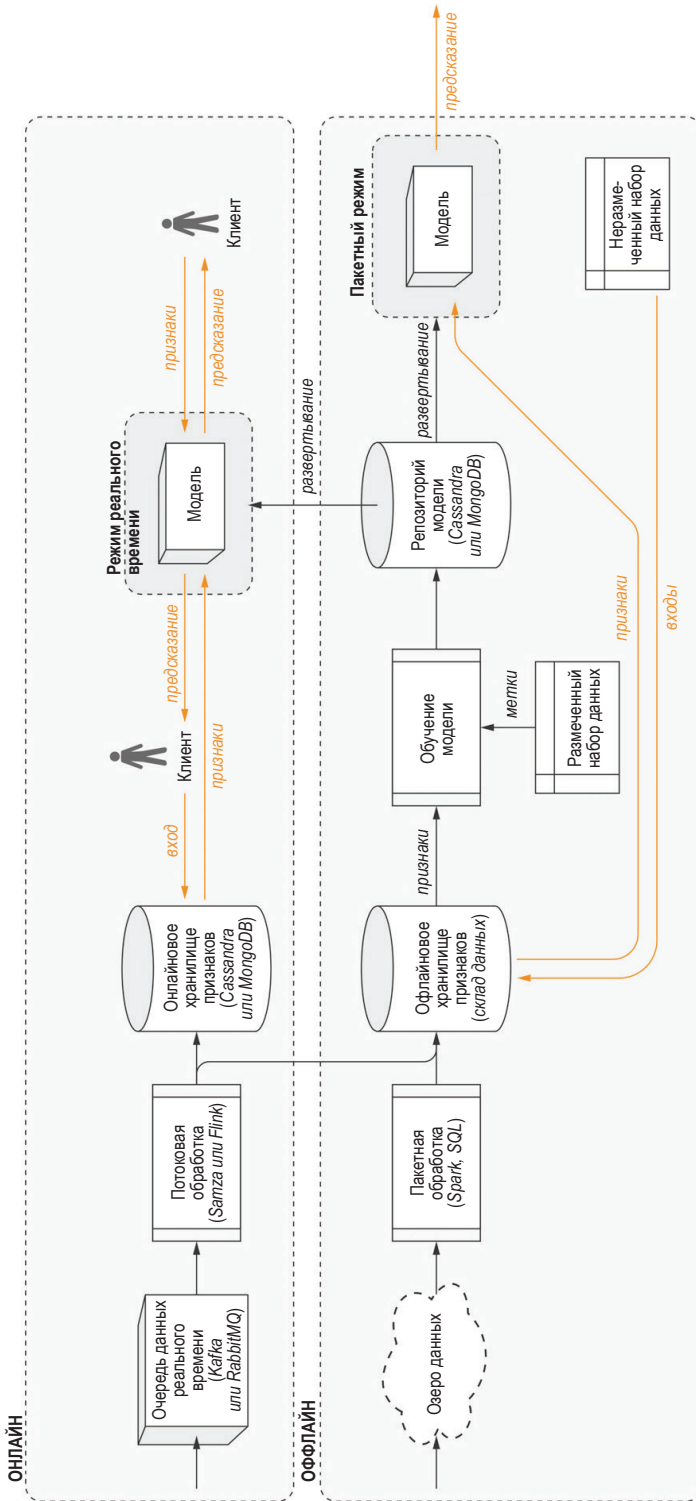


Рис. 4.21 ❖ Место хранилища признаков в конвейере машинного обучения

с использованием как значений некоторых признаков, вычисленных в онлайновом режиме, так и исторических данных из журналов и офлайновых баз данных. Примером признака, вычисляемого в онлайн-режиме, является «среднее время приготовления блюд в ресторане за последний час». В Uber офлайновое хранилище признаков синхронизируется с онлайн-режимом один или несколько раз в сутки.

4.12. РЕКОМЕНДАЦИИ ПО КОНСТРУИРОВАНИЮ ПРИЗНАКОВ

На протяжении многих лет аналитики и инженеры изобретали, экспериментально проверяли и подтверждали различные передовые практики. Сегодня они рекомендуются почти для любого проекта машинного обучения. Применение этих рекомендаций, возможно, и не приведет к значительным улучшениям, но уж точно не повредит. Выше уже была рассмотрена одна такая рекомендация – нормировать или стандартизировать признаки.

4.12.1. Генерируйте много простых признаков

В начале процесса моделирования попытайтесь сконструировать столько «простых» признаков, сколько получится. Признак считается простым, если для его кодирования не нужно много времени. Например, применение метода мешка слов к классификации документов позволяет сгенерировать тысячи признаков, написав всего пару строк кода. Если возможности оборудования позволяют, делайте любую измеримую величину признаком. Заранее не скажешь, будет ли какая-то величина полезна в сочетании с другими величинами.

4.12.2. Повторно используйте унаследованные системы

Заменяя старый алгоритм, не основанный на машинном обучении, статистической моделью, используйте выход старого алгоритма в качестве признака для новой модели. Следите за тем, чтобы старый алгоритм больше не изменялся, т. е. любые изменения могут негативно отразиться на качестве модели. Если старый алгоритм работает слишком медленно, чтобы претендовать на роль признака, используйте его входы как признаки для новой модели.

Используйте внешнюю систему как источник признаков, только если контролируете ее поведение. В противном случае владелец внешней системы, возможно, решит использовать выход вашей системы как вход для своей модели. Это создает **скрытую петлю обратной связи**, т. е. вы будете влиять на явление, на котором обучаетесь.

4.12.3. Используйте идентификаторы как признаки, когда это необходимо...

Используйте идентификаторы как признаки, когда это необходимо. Эта мысль может показаться противоречащей интуиции, поскольку уникальные идентификаторы не вносят вклад в обобщаемость. Однако использование идентификаторов позволяет создать модель, ведущую себя одним способом в общем случае и иначе в частных случаях. Например, допустим, мы хотим делать предсказания о некотором населенном пункте (городе или деревне) и располагаем свойствами этого пункта в виде признаков. Включив в состав признаков ИД населенного пункта, мы сможем добавлять обучающие примеры для одного общего места и обучить модель вести себя иначе для других конкретных мест.

Однако не используйте в качестве признаков идентификаторы примеров.

4.12.4. ...но по возможности уменьшайте количество значений

Используйте категориальные признаки с большим числом значений (больше десятка), только когда хотите, чтобы модель имела разные «режимы» поведения в зависимости от этого категориального признака. Типичные примеры – почтовый индекс или страна. Можно подумать об использовании признака «Страна», если мы хотим, чтобы модель вела себя по-разному в России и в США для похожих во всех остальных отношениях входов¹.

Если имеется категориальный признак с большим числом значений, но модель, имеющая несколько режимов, зависящих от этого признака, не нужна, попытайтесь уменьшить количество различных значений (кардинальное число признака). Сделать это можно несколькими способами. Один из них, **хеширование признаков**, мы уже рассматривали в разделе 4.2.4. Ниже кратко обсудим другие методы.

Сгруппировать похожие значения

Попытайтесь объединить некоторые значения в одну категорию. Например, если вы полагаете, что для различных населенных пунктов внутри одного региона вряд ли понадобятся разные предсказания, то сгруппируйте все почтовые индексы, относящиеся, скажем, к одной области, в единый код области. Группируйте области в более крупные образования – регионы.

Сгруппировать длинные хвосты

Аналогично попробуйте сгруппировать длинный хвост нечасто встречающихся значений в одно значение «Прочие» или объедините их с похожими частыми значениями.

¹ Часто то, что мы хотим получить от модели, и то, что диктуют данные, – «две большие разницы». Бывает, вы думаете, что модель должна давать похожие предсказания независимо от страны, а в реальности оказывается, что все не так, потому что распределение меток в обучающем наборе для разных стран различно.

Удалить признак

Если все или почти все значения категориального признака уникальны или одно значение преобладает над всеми остальными, то подумайте, нельзя ли вообще удалить этот признак.

Уменьшать кардинальное число признака следует с осторожностью. Часто бывает, что одни категориальные признаки функционально зависят от других и их предсказательная способность проистекает из употребления в комбинациях. В качестве примера возьмем штат и город. Если мы решим сгруппировать или удалить некоторые значения признака «штат», то можем случайно уничтожить информацию, которая позволяла модели отличать один город «Спрингфилд» от другого.

4.12.5. Осторожнее со счетчиками

Используйте признаки, основанные на счетчиках, с осторожностью. Некоторые счетчики с течением времени остаются приблизительно в одних и тех же границах. Например, если в мешке слов использовать счетчики лексем, а не бинарные значения, то проблемы не возникнет, коль скоро длина входного документа не растет и не уменьшается со временем. Но если имеется признак «Количество звонков с момента оформления подписки» для абонента растущего мобильного оператора, то у некоторых «старичков» количество звонков будет очень велико по сравнению с недавно пришедшими клиентами. С другой стороны, обучающие данные могли быть созданы, когда компания только пришла на рынок и «старичков» еще не было в принципе.

То же соображение применимо к ситуации, когда значения признаков распределяются по интервалам в зависимости от частоты вхождения в набор данных. Редкие сегодня значения могут стать частыми, когда будет добавлено больше данных. Рекомендуется время от времени повторно проводить оценивание модели и признаков.

4.12.6. Отбирайте признаки, когда необходимо

При необходимости производите отбор признаков. Перечислим несколько причин, когда это следует делать:

- необходимость иметь объяснимую модель (и, следовательно, сохранять наиболее значимые предикторы);
- строгие требования к оборудованию, например к объему оперативной памяти, месту на дисках и т. д.;
- краткость времени, доступного для экспериментов и (или) перестройки модели в производственной среде;
- ожидается значительный **сдвиг распределения** между двумя обучениями модели.

Если решите прибегнуть к отбору признаков, начните с Boruta.

4.12.7. Тщательно тестируйте код

Код конструирования признаков следует тщательно тестировать. Автономные тесты должны покрывать каждый экстрактор признаков. Проверяйте правильность генерирования каждого признака, используя как можно больше входных данных. Для каждого булева признака проверяйте, что он принимает значение истина, когда должен быть истинным, и значение ложь, когда должен быть ложным. Проверяйте, что числовые признаки попадают в разумный диапазон. Проверьте признаки на NaN (не число), null, нуль и пустые значения. Ошибка в экстракторе одного признака может сгубить качество всей модели. Именно с экстракторов признаков нужно начинать расследование причин странного поведения модели.

Каждый признак следует тестировать на быстродействие, потребление памяти и совместимость с производственной средой. Что хорошо работает в локальной среде, может оказаться никуда не годным после развертывания в режиме эксплуатации.

После развертывания модели в производственной среде и при каждой загрузке следует заново прогонять тесты экстракторов признаков. Если признак потребляет внешние ресурсы, например обращается к базе данных или какому-то API, может оказаться, что эти ресурсы недоступны конкретному производственному экземпляру. Экстрактор признаков должен возбуждать исключение и завершать программу, если какой-то ресурс во время выделения признаков недоступен. Избегайте «немых» отказов, которые могут оставаться незамеченными на протяжении длительного времени, из-за чего модель постепенно теряет качество или становится вообще неверной.

Также рекомендуется регулярно прогонять экстракторы признаков на фиксированных тестовых данных, проверяя, что распределение значений признаков не изменилось.

4.12.8. Синхронизируйте код, модель и данные

Версия кода выделения признаков должна быть синхронизирована с версией модели и данных, на основе которых она строилась. Все три элемента следует развертывать или откатывать одновременно. При каждой загрузке модели в производственной среде полезно проверять, что их версии совпадают.

4.12.9. Изолируйте код выделения признаков

Код выделения признаков не должен зависеть от остального кода, поддерживающего модель. Должна быть возможность обновлять код, отвечающий за каждый признак, независимо от других признаков, конвейера обработки данных или способа обращения к модели. Единственное исключение – когда много признаков генерируется массово, как при унитарном кодировании или в методе мешка слов.

4.12.10. Сериализуйте модель и экстрактор признаков совместно

По возможности совместно сериализуйте (в формате pickle в Python, RDS в R) модель и объект экстрактора признаков, который применялся во время построения модели. В производственной среде десериализуйте их вместе и используйте. По возможности избегайте использования нескольких версий кода выделения признаков.

Если в производственной среде нет возможности десериализовать одновременно модель и экстрактор признаков, используйте один и тот же код выделения при обучении модели и обращении к ней. Даже малейшее различие между кодом, с помощью которого инженер по данным обучал модель, и оптимизированным кодом, написанным отделом ИТ для производственной среды, может привести к большим ошибкам предсказания.

Подготовив код выделения признаков в производственной среде, воспользуйтесь им для переобучения модели. Всегда полностью переобучайте модель после любого изменения в коде выделения признаков.

4.12.11. Протоколируйте значения признаков

Протоколируйте значения признаков, выделенные в производственном режиме, для случайной выборки онлайн-примеров. При работе над новой версией модели эти значения пригодятся для контроля качества обучающих данных. Они позволят провести сравнение и убедиться, что значения признаков, сохраненные в производственной среде, такие же, как наблюдаемые в обучающих данных.

4.13. РЕЗЮМЕ

Признаки – это значения, выделенные из данных, с которыми будет работать модель. Каждый признак представляет некоторое свойство данных. Признаки организованы в виде векторов признаков, а модель обучается производить над векторами математические операции с целью порождения желаемого выхода.

Для текстов признаки можно генерировать массово, применяя такие методы, как мешок слов. Числа в таких векторах признаков обозначают присутствие или отсутствие определенных словарных слов в текстовом документе. Эти числа могут быть двоичными или содержать больше информации, например частоту слова в документе или величину TF-IDF.

В большинстве алгоритмов и библиотек машинного обучения требуется, чтобы все признаки были числовыми. Для преобразования категориальных признаков в числа применяются такие методы, как унитарное кодирование и кодирование средним. Если значения категориальных признаков циклические, например дни недели или часы суток, то есть способ лучше:

конвертировать циклический признак в два признака с помощью синусно-косинусного преобразования.

Хеширование признаков – способ преобразовать текстовые данные или категориальные атрибуты с большим числом значений в вектор признаков произвольной размерности. Это может быть полезно, когда унитарное кодирование или мешок слов порождает векторы признаков слишком большой размерности.

Тематическое моделирование – это семейство алгоритмов, в частности LDA и LSA, которые позволяют обучить модель преобразовывать любой документ в вектор тем требуемой размерности.

Временной ряд – это упорядоченная последовательность наблюдений. Каждое наблюдение снабжается связанным со временем атрибутом, например временная метка, дата, год и т. д. До того как нейронные сети достигли современных способностей к обучению, аналитики работали с временными рядами, применяя поверхностные средства машинного обучения. Временной ряд нужно было преобразовать в «плоские» векторы признаков.

В наше время используются нейросетевые архитектуры, адаптированные для работы с последовательностями, например LSTM, СНС и трансформеры.

Хорошие признаки имеют высокую предсказательную способность и быстро вычисляются. Кроме того, они надежные и некоррелированные.

Важно, чтобы распределение значений признаков в обучающем наборе было похоже на распределение значений, предъявляемых модели на этапе эксплуатации. Кроме того, хорошие признаки цельные, их легко понять и сопровождать. Цельность означает, что признак представляет одну простую для объяснения и понимания величину.

Чтобы увеличить предсказательную способность данных, можно синтезировать дополнительные признаки одним из следующих способов: дискретизировать существующий числовой признак, кластеризовать обучающие примеры или скомбинировать пары признаков.

Для текстов признаки можно получить из непомеченных данных в форме погружений слов или документов. В общем случае обучить погружения можно для любого типа данных, если мы сумеем сформулировать подходящую задачу предсказания и обучить глубокую модель. Затем векторы погружений извлекаются из самых правых (т. е. ближайших к выходу) полносвязных слоев.

Разумное использование методов отбора признаков позволяет исключить признаки, не дающие вклада в качество модели. Два наиболее распространенных метода – обрезание длинного хвоста и Boruta. L1-регуляризация также применима для отбора признаков.

Понижение размерности помогает получить более наглядное представление о многомерных наборах данных, а также улучшить прогностическое качество модели. В настоящее время для понижения размерности применяются такие методы, как PCA, UMAP и автокодировщики. PCA работает очень быстро, но UMAP и автокодировщики порождают более качественные визуализации.

Рекомендуется масштабировать признаки перед обучением модели, сохранять и документировать признаки в файлах схемы или хранилищах признаков, а также синхронизировать код, модель и обучающие данные.

Код выделения признаков – одна из самых важных частей системы машинного обучения. Его необходимо тщательно и систематически тестировать.

Глава 5

Обучение модели с учителем (часть 1)

Обучение модели (или моделирование) – четвертый этап жизненного цикла проекта машинного обучения.

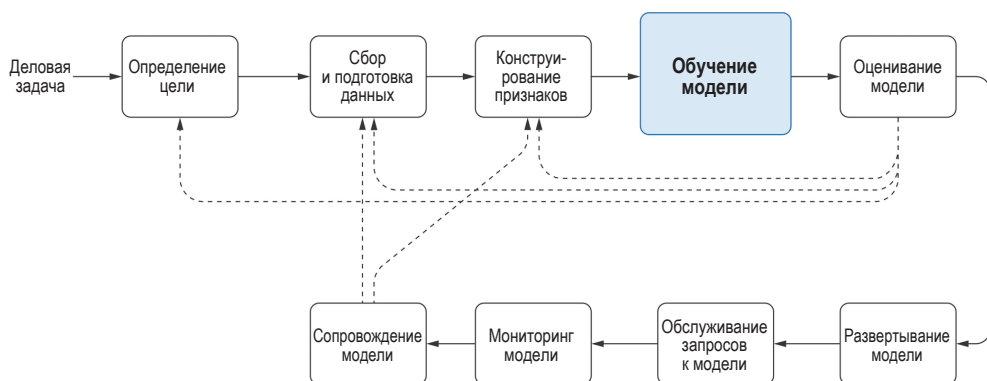


Рис. 5.1 ❖ Жизненный цикл проекта машинного обучения

Ясно, что без обучения никакую модель не построишь. Однако обучение модели – одна из самых переоцененных частей машинного обучения. В среднем инженер по машинному обучению тратит на моделирование всего 5–10 % своего времени, если вообще что-то тратит. Куда важнее успешный сбор данных, их подготовка и конструирование признаков. Обычно моделирование сводится просто к применению к данным какого-нибудь алгоритма из библиотеки `scikit-learn` или R и случайному опробованию нескольких комбинаций гиперпараметров. Поэтому если вы пропустили две предыдущие главы и сразу перескочили к моделированию, пожалуйста, вернитесь и прочитайте их. Это важно.

Как следует из названия главы, я разделил тему обучения модели с учителем на две части. В первой части мы рассмотрим подготовку к обучению, выбор алгоритма, стратегию поверхностного обучения, оценивание качества модели, компромисс между смещением и дисперсией, регуляризацию, идею конвейера машинного обучения и настройку гиперпараметров.

5.1. ПРЕЖДЕ ЧЕМ ПРИСТУПАТЬ К РАБОТЕ НАД МОДЕЛЬЮ

Прежде чем приступать к работе над моделью, необходимо проверить согласованность данных со схемой, определить достижимый уровень качества, выбрать метрику качества и принять еще несколько решений.

5.1.1. Проверка согласованности со схемой

Прежде всего убедитесь, что данные отвечают схеме, определенной в **файле схемы**. Даже если вы с самого начала подготовили данные, очень может статься, что оригинальные и текущие данные разошлись. Это расхождение можно объяснить разными факторами, перечислим наиболее вероятные:

- метод, с помощью которого данные сохранялись на жесткий диск или в базу данных, содержит ошибку;
- метод, используемый для чтения данных из того места, где они сохранены, содержит ошибку;
- кто-то мог изменить данные или схему, не поставив вас в известность.

Такие ошибки схемы необходимо выявлять и исправлять так же, как ошибки в программном коде. При необходимости всю процедуру сбора данных и конвейер обработки следует повторить с самого начала, как было описано в конце главы 3 при обсуждении **воспроизводимости**.

5.1.2. Определение достижимого уровня качества

Определение достижимого уровня качества – один из самых важных шагов. Он позволяет составить представление о том, когда стоит прекратить попытки улучшить модель. Вот несколько рекомендаций:

- если человек может пометить примеры, не прилагая чрезмерно много усилий, не применяя математику и сложные логические выкладки, то можно надеяться, что модель достигнет качества, не уступающего человеку;
- если вся информация, необходимая для принятия решения о пометке, присутствует в признаках, то можно ожидать, что ошибка будет близка к нулю;
- если вектор входных признаков содержит много сигналов (например, пикселей изображения или слов в документе), то можно ожидать, что ошибка будет близка к нулю;
- если имеется компьютерная программа для решения той же задачи классификации или регрессии, то можно ожидать, что модель будет работать не хуже. Часто качество модели машинного обучения можно улучшить в результате поступления новых помеченных данных;
- если имеется похожая, но другая система, можно ожидать похожего качества от вашей модели машинного обучения.

5.1.3. Выбор метрики качества

Об оценивании качества модели мы поговорим позже. А пока что отметим, что имеется несколько способов – метрик – оценить уровень ее качества. Не существует одной наилучшей метрики, пригодной для каждого проекта. Ее нужно выбирать, исходя из данных и конкретной задачи.

Рекомендуется выбрать одну и только одну **метрику качества**, прежде чем приступить к работе над моделью. Затем сравните различные модели и оценивайте свои успехи, используя только эту метрику.

В разделе 5.5 будет рассказано о самых популярных и удобных метриках качества модели, а также о способах комбинировать несколько метрик для получения одного числа.

5.1.4. Выбирайте правильный ориентир

Прежде чем начинать работу над прогностической моделью, важно определить уровень качества, на который можно ориентироваться. **Ориентиром** называется модель или алгоритм, который устанавливает точку отсчета для сравнения.

Зная, к чему стремиться, аналитик чувствует уверенность в том, что решение, основанное на машинном обучении, работает. Если построенная модель достигла лучшей метрики качества, чем ориентир, то применение машинного обучения оправдано.

Сравнение текущего качества модели с ориентиром помогает выбрать направление дальнейшей работы. Допустим, мы знаем, что в задаче можно достичь качества, сопоставимого с человеком. Тогда в качестве ориентира берем качество работы человека, как показано на рис. 5.2. Модель на

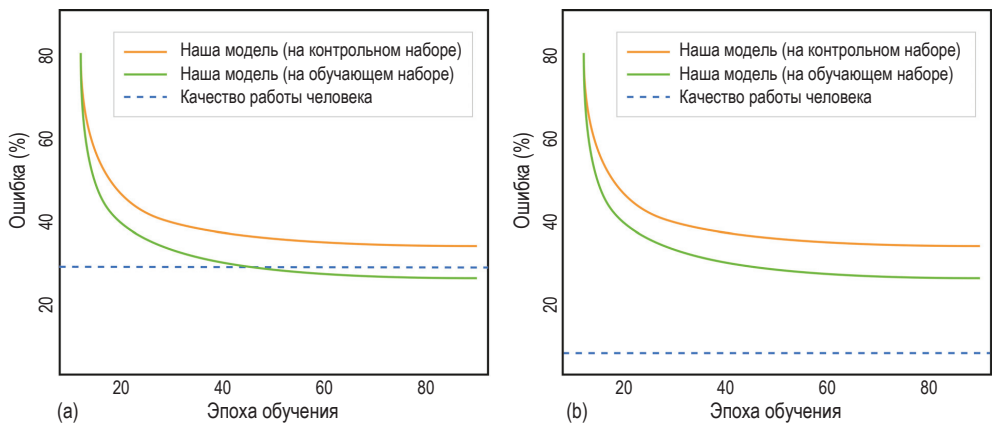


Рис. 5.2 ❖ Сравнение качества модели с качеством работы человека.

(а) Модель выглядит неплохо, поэтому мы решаем регуляризовать ее или добавить обучающие примеры; (б) модель работает плохо, поэтому нужно добавить признаки или увеличить сложность модели

рис. 5.2(a) выглядит неплохо, поэтому ее можно регуляризировать или добавить обучающие примеры. С другой стороны, модель на рис. 5.2(b) работает неважно, поэтому стоит увеличить количество признаков или повысить **сложность модели**.

Модель или алгоритм, выбранные в качестве ориентира, принимают вход и выводят опорное предсказание. Опорное предсказание должно иметь такую же природу, как предсказание модели, иначе их нельзя будет сравнивать.

Опорное предсказание не обязано быть результатом алгоритма обучения. Это может быть эвристический алгоритм, основанный на правилах, простая статистика, примененная к правилам, или что-то еще.

Два самых распространенных алгоритма-ориентира – это:

- случайное предсказание;
- правило нуля.

Алгоритм случайного предсказания дает предсказания, случайно выбирая метку из коллекции меток, сопоставленных обучающим примерам. В задаче классификации это соответствует случайному выбору одного класса из всех определенных в задаче. В задаче регрессии производится выбор из всех целевых значений, присутствующих в обучаемых данных.

Алгоритм правила нуля дает более точный опорный уровень, чем алгоритм случайного предсказания. Это значит, что он обычно улучшает метрику по сравнению со случайным предсказанием. Для получения предсказаний алгоритм правила нуля использует больше информации о задаче.

В случае классификации стратегия правила нуля состоит в том, чтобы всегда предсказывать класс, который чаще всего встречается в обучающих данных, независимо от входного значения. Это может показаться неэффективным, но давайте рассмотрим следующую задачу. Пусть обучающие данные для задачи классификации содержат 800 примеров положительного класса и 200 примеров отрицательного. Правило нуля всегда предсказывает положительный класс, а **верность** (одна из популярных метрик качества, которую мы рассмотрим в разделе 5.5.2) опорного алгоритма составляет $800/1000 = 0.8$, или 80 %, что не так плохо для такого простого классификатора. Теперь мы знаем, что наша статистическая модель, не важно, насколько она близка к оптимальной, должна обеспечить верность не менее 80 %.

Рассмотрим правило нуля для регрессии. Согласно этому алгоритму стратегия регрессии состоит в том, чтобы предсказывать выборочное среднее целевых значений, наблюдаемых в обучающих данных. Скорее всего, такая стратегия будет давать меньшую частоту ошибок, чем случайное предсказание.

Если вы работаете над стандартной, так называемой классической задачей предсказания, то можете использовать какой-нибудь из лучших на сегодняшний день алгоритмов, имеющихся в популярных библиотеках, скажем `scikit-learn` на Python. Например, в случае классификации текста представьте текст в виде **мешка слов**, а затем обучите модель **опорных векторов с линейным ядром**. Затем попробуйте превзойти ее результат, предложив свой собственный продвинутый метод. Этот подход должен хорошо работать для классификации изображений, машинного перевода и других хорошо изученных эталонных задач.

Для числового набора данных общего вида достойной опорой может стать линейная модель, например линейная или логистическая регрессия либо метод **k ближайших соседей** с $k = 5$. Для классификации изображений подойдет простая **сверточная нейронная сеть** (СНС) с тремя сверточными слоями (32 – 64 – 32 блока в слое, и за каждым сверточным слоем следует слой тахпулинга и слой прореживания) и двумя полносвязными слоями в конце (в одном 128 блоков, а число блоков в другом соответствует числу желаемых выходов).

Можно также использовать какую-нибудь существующую систему на основе правил или создать собственную систему. Например, если задача заключается в том, чтобы построить модель, которая предсказывает, понравится ли посетителю сайта рекомендованная статья, то простая система на основе правил могла бы работать следующим образом. Возьмем все статьи, которые нравились пользователю, найдем в этих статьях десять самых частых слов (с максимальной оценкой **TF-IDF**) и предскажем, что статья понравится пользователю, если в ней встречается по крайней мере пять из этих десяти слов. Кроме того, в сети нет недостатка в специализированных библиотеках машинного обучения и API. Если их можно использовать непосредственно или адаптировать для решения вашей задачи, то почему бы не рассматривать их как ориентир.

Точно определить опорный уровень, достижимый человеком, не всегда просто. Можно воспользоваться службой **механического турка** (МТ), предоставляемой компанией Amazon. Это веб-платформа, на которой люди решают простые задачи за вознаграждение. МТ предлагает API, который можно вызывать для получения предсказаний от человека. Качество таких предсказаний может варьироваться от очень низкого до сравнительно высокого в зависимости от задачи и величины вознаграждения. МТ обходится относительно недорого, так что предсказания можно получить быстро и в большом количестве.

Для повышения качества предсказаний, данных туркерами (так называют людей по ту сторону механического турка от вас), некоторые аналитики используют **ансамбль туркеров**. Можно попросить трех или пятерых туркеров пометить один и тот же пример, а затем выбрать класс, назначенный большинством голосов (или в случае регрессии усреднить метки). Более дорогая альтернатива – обратиться к профильным специалистам (или их ансамблю, для повышения качества) с просьбой пометить ваши данные.

5.1.5. Разбиение данных на три набора

Напомним, что в общем случае для построения солидной модели нужно три набора данных. Первый, **обучающий набор** используется для обучения модели. Это те данные, которые «видит» алгоритм машинного обучения. Второй и третий наборы зарезервированы. **Контрольный набор** не виден алгоритму обучения. Аналитик использует его для оценки качества различных алгоритмов (или одного алгоритма, но с разными значениями гиперпараметров) или моделей применительно к новым данным. Оставшийся **тестовый набор** тоже не виден алгоритму обучения, он используется в конце проекта для оценивания и документирования качества той модели, которая наилучшим образом показала себя на контрольных данных.

Процесс разбиения всего набора данных на три части описан в разделе 3.6. Здесь я только повторю два самых важных его свойства.

1. Контрольный и тестовый наборы должны быть выбраны из одного и того же статистического распределения. То есть их свойства должны быть максимально похожи, но сами примеры, очевидно, должны быть различны и в идеале получены независимо друг от друга.
2. Выбирайте контрольные и тестовые данные из распределения, выглядящего максимально похоже на данные, которые вы ожидаете увидеть после развертывания модели в производственной среде. Оно может отличаться от распределения обучающих данных.

Два слова по поводу последнего пункта. Чаще всего аналитики просто перемешивают весь набор данных, а затем случайным образом выбирают из него три набора. Но на практике нередко встречается много примеров, не похожих на производственные данные. Иногда этих примеров в изобилии и (или) они недорого обходятся. Если использовать такие данные в проекте, то может иметь место **сдвиг распределения**, о котором аналитик может знать, а может и не знать.

Если вы знаете о сдвиге распределения, то имеет смысл поместить все легкодоступные примеры в обучающий набор, но не использовать их в контрольном и тестовом наборах. Тогда вы сможете оценить модель на данных, похожих на реальные. В противном случае при тестировании модели может получиться чрезмерно оптимистичная оценка метрики качества и будет выбрана неоптимальная модель.

Сдвиг распределения может оказаться нетривиальной проблемой. Использование другого распределения для обучения может быть сознательным выбором, продиктованным трудностями получения данных. Но аналитик может и не знать, что статистические свойства обучающих и реальных данных различны. Так часто бывает, когда модель редко обновляется после развертывания в производственном режиме, а в обучающий набор добавляются новые примеры. Свойства данных, использованных при обучении модели и при ее контроле и тестировании, со временем могут разойтись. В разделе 6.3 следующей главы даны рекомендации по решению этой проблемы.

5.1.6. Предварительные условия для обучения с учителем

Прежде чем начинать работу над моделью, убедитесь, что выполнены следующие условия.

1. Имеется размеченный набор данных.
2. Набор данных разделен на три поднабора: обучающий, контрольный и тестовый.
3. Примеры в контрольном и тестовом наборах статистически похожи.
4. Вы конструировали признаки и заполняли отсутствующие значения, пользуясь только обучающими данными.

5. Все примеры преобразованы в числовые векторы признаков¹.
6. Выбрана метрика качества, возвращающая одно число (см. раздел 5.5).
7. Определен ориентир.

5.2. ПРЕДСТАВЛЕНИЕ МЕТОК ДЛЯ МАШИННОГО ОБУЧЕНИЯ

В классической постановке метки являются категориальными признаками. Например, при классификации изображений возможны метки «кошка», «собака», «автомобиль», «здание» и т. д.

Некоторые алгоритмы машинного обучения, в частности включенные в `scikit-learn`, принимают метки в их естественной форме: в виде строк. Библиотека сама преобразовывает строки в числа, которые могут быть поданы на вход конкретному алгоритму обучения.

Но некоторые реализации, например нейронных сетей, требуют, чтобы преобразование меток в числа выполнил аналитик.

5.2.1. Многоклассовая классификация

В случае **многоклассовой классификации** (когда модель предсказывает только одну метку по входному вектору признаков) для преобразования меток в бинарные векторы обычно используется **унитарное кодирование**. Например, пусть определены классы {собака, кошка, другое} и имеются следующие данные:

Изображение	Метка
image_1.jpg	собака
image_2.jpg	собака
image_3.jpg	кошка
image_4.jpg	другое
image_5.jpg	кошка

При унитарном кодировании были бы сгенерированы такие бинарные векторы классов:

```
dog = [1, 0, 0],
cat = [0, 1, 0],
other = [0, 0, 1].
```

¹ Как отмечалось в предыдущей главе, большинство современных библиотек и пакетов машинного обучения ожидают получить числовые векторы признаков. Но некоторые алгоритмы, например решающие деревья, могут естественно работать с категориальными признаками.

После преобразования категориальных меток в бинарные векторы данные принимают вид:

Изображение	Метка
image_1.jpg	[1,0,0]
image_2.jpg	[1,0,0]
image_3.jpg	[0,1,0]
image_4.jpg	[0,0,1]
image_5.jpg	[0,1,0]

5.2.2. Многозначная классификация

В случае **многозначной классификации** модель может предсказывать сразу несколько меток для одного входа (например, на картинке могут быть изображены и собака, и кошка). Тогда для представления меток, назначенных каждому примеру, можно использовать **мешок слов**. Допустим, имеются следующие данные:

Изображение	Метки
image_1.jpg	собака, кошка
image_2.jpg	собака
image_3.jpg	кошка, другое
image_4.jpg	другое
image_5.jpg	кошка, собака

После преобразования меток в бинарные векторы данные принимают вид:

Изображение	Метки
image_1.jpg	[1,1,0]
image_2.jpg	[1,0,0]
image_3.jpg	[0,1,1]
image_4.jpg	[0,0,1]
image_5.jpg	[1,1,0]

Ознакомьтесь с документацией по конкретной реализации алгоритма обучения, чтобы узнать, в каком формате он принимает входные данные.

5.3. Выбор алгоритма обучения

Выбор алгоритма машинного обучения может оказаться трудной задачей. Если у вас достаточно времени, можете попробовать все. Но обычно время, отведенное на решение задачи, ограничено. Чтобы сделать обоснованный выбор, задайте себе несколько вопросов, перед тем как приступить к работе. В зависимости от ответов можно составить короткий список потенциальных алгоритмов и попробовать применить их к данным.

5.3.1. Основные свойства алгоритма обучения

Ниже приведено несколько вопросов и ответов, которыми можно руководствоваться при выборе алгоритма или модели машинного обучения.

Объяснимость

Нужно ли объяснять предсказания модели технически не подготовленной аудитории? Самые верные алгоритмы и модели машинного обучения являются «черными ящиками». Они делают очень мало ошибок предсказания, но трудно понять, а еще труднее объяснить, почему было дано именно такое предсказание. Примерами могут служить **глубокие нейронные модели** и **ансамблевые модели**.

С другой стороны, алгоритмы **k ближайших соседей**, **линейной регрессии** и **обучения решающих деревьев** не всегда самые верные. Однако для интерпретации их предсказаний даже не нужно быть экспертом.

В памяти или вне памяти

Можно ли загрузить весь набор данных в оперативную память вашего ноутбука или сервера? Если да, то доступен широкий спектр алгоритмов. В противном случае придется ограничиться **алгоритмами инкрементного обучения**, которые умеют улучшать модель, читая данные постепенно. Примерами могут служить **наивный байесовский** классификатор и алгоритмы обучения нейронных сетей.

Число признаков и примеров

Сколько обучающих примеров имеется в вашем наборе данных? Сколько признаков у каждого примера? Некоторые алгоритмы, в частности применяемые для обучения нейронных сетей и случайных лесов, могут обрабатывать огромное количество примеров и миллионы признаков. Другие, например метод опорных векторов (SVM), имеют сравнительно скромную емкость.

Нелинейность данных

Допускают ли данные линейное разделение? Можно ли для их моделирования применить линейную модель? Если да, то неплохими кандидатами будут SVM с линейным ядром, линейная и логистическая регрессии. В противном случае глубокие нейронные сети и ансамблевые модели работают лучше.

Скорость обучения

Сколько времени разрешено потратить алгоритму обучения на построение модели и как часто придется переобучать модель на изменившихся данных? Если обучение занимает два дня и модель требуется переобучать каждые 4 часа, то, очевидно, она никогда не будет актуальной. Нейронные сети обучаются медленно. Простые алгоритмы типа линейной и логистической регрессий или решающих деревьев – гораздо быстрее.

В специализированных библиотеках есть очень эффективные реализации некоторых алгоритмов. Можете потратить время на поиск таких библиотек. Некоторые алгоритмы, например обучения случайных лесов, выигрывают от наличия нескольких процессорных ядер, поэтому время их обуче-

ния можно значительно уменьшить на машине с несколькими десятками ядер. Кроме того, некоторые библиотеки задействуют GPU (графические процессоры) для ускорения обучения.

Скорость предсказания

Как быстро модель должна генерировать предсказания? Будет ли она эксплуатироваться в среде, где требуется очень высокая пропускная способность? Такие модели, как SVM, линейная и логистическая регрессии и не слишком глубокие нейронные сети прямого распространения, генерируют предсказания очень быстро. Другие модели, например kNN, ансамблевые алгоритмы и очень глубокие или рекуррентные нейронные сети, работают медленнее.

Если вы не хотите гадать, какой алгоритм будет наилучшим в вашем случае, то можно применить популярный способ: протестировать несколько кандидатов на **контрольном наборе**, считая выбор алгоритма гиперпараметром. О настройке гиперпараметров мы будем говорить в разделе 5.6.

5.3.2. Выборочная проверка алгоритмов

Формирование короткого списка алгоритмов-кандидатов для данной задачи иногда называют **выборочной проверкой алгоритмов** (algorithm spot-checking). Чтобы выборочная проверка была максимально эффективной, рекомендуется:

- выбирать алгоритмы, основанные на разных принципах (иногда их называют ортогональными), например на основе экземпляров, на основе ядра, поверхностного обучения, глубокого обучения, ансамблевые;
- пробовать каждый алгоритм с 3–5 разными значениями наиболее чувствительных гиперпараметров (например, количество соседей k в алгоритме k ближайших соседей, штраф C в методе опорных векторов или порог принятия решения в логистической регрессии);
- использовать во всех экспериментах одно и то же разделение на обучающий и контрольный наборы;
- если алгоритм обучения недетерминированный (например, алгоритмы обучения нейронных сетей и случайных лесов), то выполнять несколько экспериментов, а затем усреднять результаты;
- по завершении проекта зафиксировать, какие алгоритмы оказались наилучшими, и использовать эту информацию при работе над похожими задачами в будущем.

Пока вы еще не изучили задачу как следует, пытайтесь решить ее, используя как можно больше ортогональных подходов, а не тратьте слишком много времени на самый многообещающий подход. Обычно лучше потратить время на эксперименты с новыми алгоритмами и библиотеками, чем стараться выжать максимум из того, с которым вы лучше знакомы.

Если у вас нет времени на скрупулезную выборочную проверку алгоритмов, то просто поищите эффективную реализацию алгоритма или модели обучения, которая, согласно большинству современных статей, демонстрирует наилучшее качество на задачах, сходных с вашей, и используйте ее для решения своей задачи. Если вы пользуетесь библиотекой scikit-learn, то можете прибегнуть к диаграмме выбора алгоритмов на рис. 5.3.

5.4. ПОСТРОЕНИЕ КОНВЕЙЕРА

Многие современные пакеты и каркасы машинного обучения поддерживают понятие **конвейера**. Конвейер – это последовательность преобразований, которым подвергаются обучающие данные, прежде чем станут моделью. Пример конвейера для обучения модели классификации на коллекции помеченных текстовых документов приведен ниже.

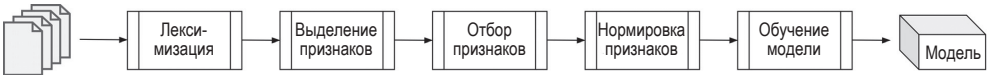


Рис. 5.4 ❖ Конвейер для порождения модели, начиная с первичных данных

Каждая стадия конвейера, кроме первой, получает вход от предыдущей стадии. На вход первой стадии подается обучающий набор данных.

Ниже приведен фрагмент кода на Python, в котором конструируется простой конвейер **scikit-learn**. Он состоит из двух шагов: 1) понижение размерности **методом главных компонент** (PCA) и 2) обучение классификатора **методом опорных векторов** (SVM).

```

1 from sklearn.pipeline import Pipeline
2 from sklearn.svm import SVC
3 from sklearn.decomposition import PCA
4
5 # Определить конвейер
6 pipe = Pipeline([('dim_reduction', PCA()), ('model_training', SVC())])
7
8 # Обучить параметры PCA и SVC
9 pipe.fit(X, y)
10
11 # Сделать предсказание
12 pipe.predict(new_example)
  
```

При выполнении команды `pipe.predict(new_example)` первый пример сначала преобразуется в вектор пониженной размерности с помощью модели PCA. Этот вектор подается на вход модели SVM. Модели PCA и SVM были обучены одна за другой в процессе выполнения команды `pipe.fit(X, y)`.

К сожалению, определение и обучение конвейеров на R устроено не так просто, как в Python, поэтому я не включил его в книгу.

Конвейер можно сохранить в файле, это похоже на сохранение модели. Затем он развертывается в производственной среде и используется для совершения предсказаний. Иными словами, в процессе **вычисления оценки** входной пример проходит по конвейеру и становится выходом.

Как видим, понятие конвейера является обобщением понятия модели. Начиная с этого момента, если явно не оговорено противное, я, говоря об обучении, сохранении, развертывании, использовании, мониторинге и сопровождении модели, буду иметь в виду весь конвейер.

Прежде чем переходить к проблеме обучения модели, нужно решить, как измерять ее качество. Часто имеется выбор между несколькими конкурирующими моделями-кандидатами, но в производственном режиме должна быть развернута только одна.

5.5. ОЦЕНИВАНИЕ КАЧЕСТВА МОДЕЛИ

Напомним, что в состав **зарезервированных данных** входят примеры, которые не предъявлялись алгоритму на этапе обучения. Если модель хорошо работает на зарезервированном наборе, то говорят, что она **хорошо обобщается**, что модель хорошего качества или просто хорошая. Самый распространенный способ получить хорошую модель – сравнить разные модели, вычислив **метрику качества** на зарезервированных данных.

5.5.1. Метрики качества для регрессии

Для оценки моделей классификации и регрессии применяются разные метрики. Сначала рассмотрим метрики качества для регрессии: среднеквадратическую ошибку, медианное абсолютное отклонение и частоту почти правильных предсказаний.

Метрика, которая чаще всего используется для оценки качества модели регрессии, совпадает с **функцией стоимости**; это **среднеквадратическая ошибка** (СКО, *англ.* MSE), определенная следующим образом:

$$\text{MSE}(f) \stackrel{\text{def}}{=} \frac{1}{N} \sum_{i=1 \dots N} (f(\mathbf{x}_i) - y_i)^2, \quad (5.1)$$

где f – модель, которая принимает вектор признаков \mathbf{x} на входе и выводит предсказание, а $i = 1, \dots, N$ – индекс примера в наборе данных.

Хорошо аппроксимирующая модель регрессии предсказывает значения, близкие к наблюдаемым данным. **Модель среднего**, которая всегда предсказывает среднее арифметическое меток обучающих данных, можно было бы использовать в отсутствие информативных признаков. Поэтому модель регрессии должна аппроксимировать лучше, чем модель среднего. Таким образом, модель среднего является **ориентиром**. Если СКО модели регрессии больше, чем СКО ориентира, то модель регрессии, очевидно, испытывает проблемы. Она может быть **переобучена** или **недообучена** (мы рассмотрим эти понятия в разделе 5.8). Возможно также, что задача была поставлена некорректно или программный код содержит ошибку.

Если данные содержат выбросы, т. е. примеры, отстоящие очень далеко от «истинной» прямой регрессии, то это может сильно повлиять на величину СКО. По определению, квадратичная ошибка для таких примеров будет велика. В этом случае лучше использовать другую метрику, **медианное абсолютное отклонение** (median absolute error – MdAE):

$$\text{MdAE} \stackrel{\text{def}}{=} \text{median}(\{|f(\mathbf{x}_i) - y_i|\}_{i=1}^N),$$

где $\{|f(\mathbf{x}_i) - y_i|\}_{i=1}^N$ – множество абсолютных отклонений для всех примеров для $i = 1$ до N , на которых производится оценивание модели.

Частота почти правильных предсказаний (almost correct predictions error rate – ACPER) – это процентная доля предсказаний, отклоняющихся от истинного значения не более чем на процент p . Для вычисления ACPER нужно действовать следующим образом.

1. Определить пороговое значение процентной ошибки, которое мы готовы считать допустимым (например, 2 %).
2. Для каждого истинного значения целевого показателя y_i предсказание должно быть заключено между $y_i + 0.02y_i$ и $y_i - 0.02y_i$.
3. Используя все примеры для $i = 1, \dots, N$, вычислить процентную долю предсказанных значений, удовлетворяющих сформулированному выше правилу. Это и будет метрика ACPER для вашей модели.

5.5.2. Метрики качества для классификации

В случае классификации ситуация немного сложнее. Самыми распространенными метриками являются:

- точность-полнота;
- верность;
- верность с учетом стоимости;
- площадь под ROC-кривой (AUC).

Для простоты я буду иллюстрировать эти метрики на задаче бинарной классификации. А там, где необходимо, покажу, как обобщить подход на случай более двух классов.

Сначала разберемся с матрицей неточностей.

Матрицей неточностей называется таблица, из которой видно, насколько успешно модель классификации предсказывает принадлежность примеров к различным классам. По одной оси располагаются классы, предсказываемые моделью, а по другой – фактические метки. Предположим, что модель предсказывает классы «спам» и «не спам».

	Спам (предсказано)	Не спам (предсказано)
Спам (фактически)	23 (TP)	1 (FN)
Не спам (фактически)	12 (FP)	556 (TN)

Из этой матрицы видно, что из 24 фактических примеров спама модель правильно классифицировала 23. В этом случае мы говорим, что имеется **23 истинно положительных** (true positive) результата, или $TP = 23$. Модель неправильно классифицировала 1 пример спама как не спам. Следовательно, мы имеем **1 ложноотрицательный** (false negative) результат, или $FN = 1$. Аналогично из 568 фактических примеров не спама модель правильно классифицировала 556 и неправильно 12 примеров (**556 истинно отрицательных** результатов, $TN = 556$, и **12 ложноположительных** результатов, $FP = 12$).

В матрице неточностей для многоклассовой классификации число строк и столбцов равно числу различных классов. Она иногда позволяет определить закономерности ошибок. Например, она могла бы показать, что модель, обученная распознавать разные виды животных, часто ошибочно предсказывает «кошку» вместо «пантеры» или «мышь» вместо «крысы». В таком случае можно добавить дополнительные примеры этих видов, чтобы алгоритм обучения «увидел» разницу между ними. Или же можно было бы добавить признаки, которые помогут алгоритму лучше различать такие пары.

Матрица неточностей используется для вычисления трех метрик качества: точности, полноты и верности. Точность и полнота чаще всего используются для оценивания бинарной модели.

Точностью (precision) называется отношение истинно положительных предсказаний к общему числу положительных предсказаний:

$$\text{precision} \stackrel{\text{def}}{=} \frac{TP}{TP + FP}.$$

Полнотой (recall) называется отношение истинно положительных предсказаний к общему числу положительных примеров:

$$\text{recall} \stackrel{\text{def}}{=} \frac{TP}{TP + FN}.$$

Чтобы понять смысл и важность точности и полноты для оценивания модели, полезно представить себе задачу предсказания как задачу поиска в базе данных документов, отвечающих запросу. Точность – это доля релевантных документов в списке всех возвращенных. А полнота – это отношение числа фактически возвращенных релевантных документов к полному числу релевантных документов, которое следовало бы вернуть.

При обнаружении спама мы хотим, чтобы точность была высокой, тогда хорошие сообщения не окажутся в папке спама. Мы готовы смириться с низкой полнотой, т. к. можем разобраться с немногочисленными спамными сообщениями в папке входящих.

На практике мы должны выбирать между высокой точностью и высокой полнотой. Иметь то и другое одновременно невозможно. Это называется **компромиссом между точностью и полнотой**. Достичь этой цели можно разными способами:

- назначать более высокий вес примерам из определенного класса. Например, алгоритм SVM в библиотеке scikit принимает веса классов в качестве входных параметров;
- настроить гиперпараметры, так чтобы максимизировать либо точность, либо полноту на контрольном наборе;
- варьировать порог принятия решений в алгоритмах, возвращающих оценки предсказаний. Предположим, что имеется модель логистической регрессии или решающее дерево. Чтобы увеличить точность (ценой уменьшения полноты), мы можем решить, что предсказание будет положительным, только если возвращенная моделью оценка больше 0.9 (место подразумеваемого по умолчанию значения 0.5).

Хотя точность и полнота определены для бинарной классификации, их можно использовать и для оценивания модели **многоклассовой классификации**. Сначала выберем класс, для которого хотим оценить эти метрики. Затем будем считать все примеры из этого класса положительными, а все примеры из остальных классов отрицательными.

На практике для сравнения двух моделей желательно использовать только один показатель качества. Так, хотелось бы избежать ситуаций, когда первая модель имеет высокую точность, а вторая – высокую полноту: как в таком случае решить, какая модель лучше? Эта техника называется **оптимизацией при разумной достаточности** (optimizing and satisficing).

Некоторые специалисты используют комбинацию точности и полноты, которая называется **F-мерой**. Традиционная F-мера, или F_1 -оценка, равна среднему гармоническому точности и полноты

$$F_1 = \left(\frac{2}{\text{recall}^{-1} + \text{precision}^{-1}} \right) = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}}.$$

Вообще же F-мера параметризуется положительным вещественным числом β , выбранным так, что полнота считается в β раз важнее точности:

$$F_\beta = (1 + \beta^2) \times \frac{\text{precision} \times \text{recall}}{(\beta^2 \times \text{precision}) + \text{recall}}.$$

Часто используются значения β , равные 2 (полнота в два раза важнее точности) и 0.5 (точность в два раза важнее полноты).

Найдите такую комбинацию обеих метрик, которая лучше всего работает в вашей задаче. Помимо F-меры, есть и другие способы скомбинировать одно число из нескольких метрик:

- простое или взвешенное среднее метрик;
- задать порог допустимости для $n - 1$ метрик и оптимизировать n -ю (обобщение описанной выше техники оптимизации с разумной достаточностью);
- придумать собственный «рецепт», учитывающий специфику задачи.

Верность (ассигасу) – это число правильно классифицированных примеров, поделенное на общее число классифицированных примеров. В терминах матрицы неточностей верность равна

$$\text{accuracy} \stackrel{\text{def}}{=} \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \quad (5.2)$$

Верность полезна, когда ошибки предсказания всех классов считаются одинаково важными. Так, например, обстоит дело в задаче распознавания объектов домашним роботом: стул ничуть не важнее стола. Но в задаче различения спама и не спама это уже не так. Надо полагать, что вы скорее будете готовы смириться с ложноотрицательными результатами, чем с ложноположительными. Напомним, что ложноположительный результат в этом контексте – это когда модель помещает письмо от друга в папку спама и вы

его не видите. Ложноотрицательный результат – когда спамное сообщение помещается в папку входящих – не такая досадная ситуация.

Чтобы разобраться с ситуациями, когда классы имеют различную важность, полезна метрика **верности с учетом стоимости**. Сначала назначим стоимость (положительное число) ошибкам обоих типов: FP и FN. Затем вычислим счетчики TP, TN, FP, FN как обычно и умножим счетчики FP и FN на соответствующие стоимости, перед тем как вычислять верность по формуле (5.2).

Верность измеряет качество модели для всех классов сразу и возвращает одно число, что очень удобно. Однако верность не может служить хорошей метрикой качества, если данные не сбалансированы. В **несбалансированном наборе данных** примеры, принадлежащие одному или нескольким классам, составляют подавляющее большинство, тогда как для других классов имеется очень мало примеров. Несбалансированность данных может оказать существенное негативное влияние на модель. Как с этим бороться, мы поговорим в разделе 6.4.

Для несбалансированных данных лучше использовать **среднюю верность** (per-class accuracy). Сначала вычислим верность предсказания для каждого класса $\{1, \dots, C\}$, а затем возьмем среднее C отдельных мер верности. Для приведенной выше матрицы неточностей в задаче распознавания спама верность для класса «спам» равна $23/(23 + 1) = 0.96$, а верность для класса «не спам» – $556/(12 + 556) = 0.98$. Следовательно, средняя верность равна $(0.96 + 0.98)/2 = 0.97$.

Средняя верность – неподходящая мера качества модели для задач многоклассовой классификации, если многие классы представлены совсем небольшим количеством примеров (грубо говоря, меньше десятка на класс). В таком случае значения верности, полученные для задач бинарной классификации, соответствующих этим миноритарным классам, будут статистически ненадежны.

Каппа Коэна – это метрика качества, применимая как к многоклассовым, так и к несбалансированным задачам. Ее преимущество перед верностью в том, что каппа Коэна говорит, насколько построенная модель работает лучше классификатора, который случайным образом предсказывает класс, исходя из частоты классов.

Каппа Коэна определяется следующим образом:

$$\kappa \stackrel{\text{def}}{=} \frac{p_o - p_e}{1 - p_e},$$

где p_o называется наблюдаемым согласием, а p_e – ожидаемым согласием.

Снова рассмотрим матрицу неточностей:

	Класс1 (предсказано)	Класс2 (предсказано)
Класс1 (фактически)	a	b
Класс2 (фактически)	c	d

Из этой матрицы получаем наблюдаемое согласие p_o :

$$p_o \stackrel{\text{def}}{=} \frac{a + d}{a + b + c + d}.$$

А ожидаемое согласие p_e равно

$$p_{\text{class1}} \stackrel{\text{def}}{=} \frac{a + b}{a + b + c + d} \times \frac{a + c}{a + b + c + d}$$

и

$$p_{\text{class2}} \stackrel{\text{def}}{=} \frac{c + d}{a + b + c + d} \times \frac{b + d}{a + b + c + d}.$$

Значение каппы Коэна всегда меньше или равно 1. Значение 0 или меньше означает, что у модели имеются проблемы. Хотя не существует универсального способа интерпретации каппы Коэна, обычно считается, что значения между 0.61 и 0.80 свидетельствуют о хорошем качестве модели, а 0.81 и выше – об очень хорошем.

ROC-кривая (термин ROC, «receiver operating characteristic» – рабочая характеристика приемника – пришел из радиолокационной техники) – распространенный метод оценивания моделей классификации. При построении ROC-кривых используется комбинация **частоты истинно положительных результатов (полноты)** и **частоты ложноположительных результатов** (доли неправильно предсказанных отрицательных примеров), чтобы составить представление о качестве классификации.

Частота истинно положительных результатов (TPR) и частота ложноположительных результатов (FPR) определяются соответственно как

$$\text{TPR} \stackrel{\text{def}}{=} \frac{\text{TP}}{\text{TP} + \text{FN}} \quad \text{и} \quad \text{FPR} \stackrel{\text{def}}{=} \frac{\text{FP}}{\text{FP} + \text{TN}}.$$

ROC-кривые можно использовать только для оценивания классификаторов, возвращающих оценку (или вероятность) предсказания. Например, с их помощью можно оценивать логистическую регрессию, нейронные сети и решающие деревья (и ансамбли моделей на основе решающих деревьев).

Для построения ROC-кривой нужно сначала дискретизировать диапазон оценок. Например, диапазон $[0, 1]$ можно дискретизировать так: $[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]$. Затем каждое дискретное значение используется как порог предсказания. Так, чтобы вычислить TPR и FPR для порога 0.7, мы применяем модель к каждому примеру и получаем оценку. Если оценка больше или равна 0.7, то предсказывается положительный класс, иначе отрицательный.

Взгляните на рис. 5.5. Легко видеть, что если порог равен 0, то все наши предсказания положительны, поэтому TPR и FPR равны 1 (правый верхний угол). С другой стороны, если порог равен 1, то положительное предсказание вообще невозможно. Тогда TPR и FPR равны 0, что соответствует левому нижнему углу.

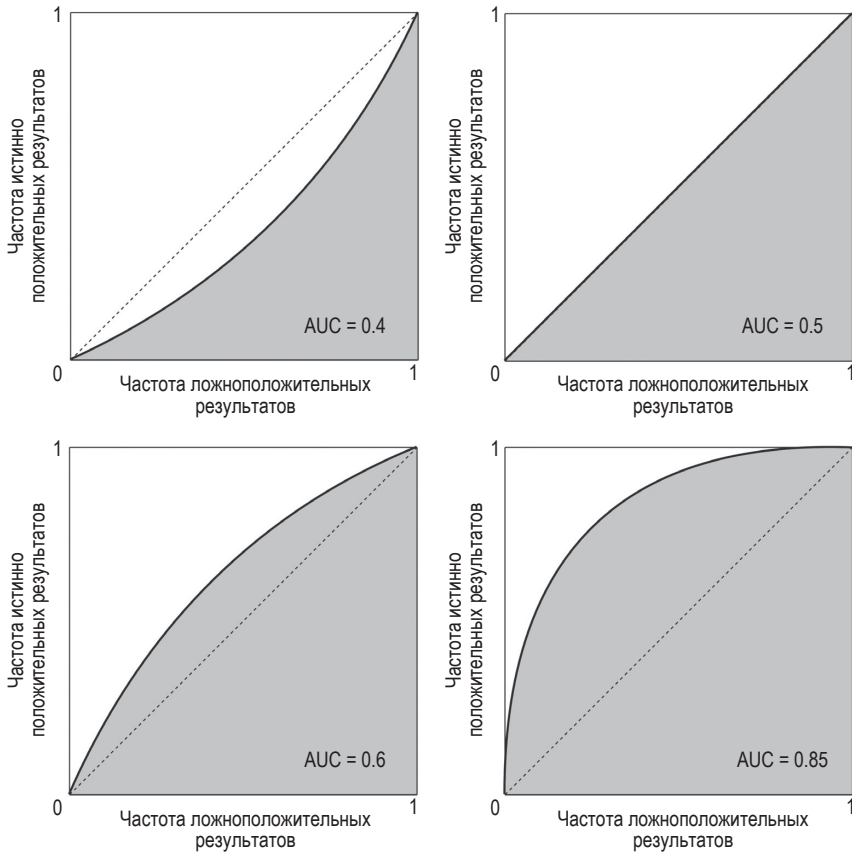


Рис. 5.5 ❖ Площадь под ROC-кривой (закрашено серым)

Чем больше **площадь под ROC-кривой** (area under curve – **AUC**), тем классификатор лучше. Классификатор с AUC больше 0.5 лучше модели, дающей случайные предсказания. Если AUC меньше 0.5, то определенно имеется какая-то проблема, чаще всего ошибка в коде или неправильно размеченные данные. Для идеального классификатора AUC равна 1. На практике для получения хорошего классификатора нужно выбрать пороговое значение, которое дает TPR, близкую к 1, при этом FPR недалеко отклоняется от 0.

ROC-кривые популярны, потому что их сравнительно легко понять. Они улавливают более одного аспекта классификации, т. к. принимают во внимание частоты как ложноположительных, так и ложноотрицательных результатов. Они позволяют аналитику просто и наглядно сравнивать качество различных моделей.

5.5.3. Метрики качества для ранжирования

Точность и полнота естественно применимы к задаче ранжирования. Напомним, что эти метрики удобно интерпретировать как качество результатов

поиска документов. Точность – это доля найденных релевантных документов в списке всех возвращенных документов. А полнота – это отношение количества возвращенных релевантных документов к общему числу релевантных документов, которые должны были быть найдены.

Недостаток измерения качества моделей ранжирования в терминах точности и полноты заключается в том, что эти метрики трактуют все найденные документы одинаково. Релевантный документ, поставленный в позицию k , не более и не менее релевантен, чем тот, что оказался в начале списка. Но не этого мы обычно хотим от системы поиска документов. Для человека, который смотрит на результаты поиска, первые несколько результатов имеют больший вес, чем результаты в конце списка.

Дисконтированный накопительный выигрыш (discounted cumulative gain – DCG) – популярная мера ранжирования качества в поисковых системах. DCG измеряет полезность, или выигрыш документа, на основе его позиции в списке результатов. Выигрыш суммируется нарастающим итогом с начала до конца списка, при этом чем ниже позиция результата, тем сильнее обесценивается (дисконтируется) его выигрыш.

Чтобы лучше понять смысл дисконтированного накопительного выигрыша, введем еще одну меру – накопительный выигрыш.

Накопительным выигрышем (cumulative gain – CG) называется сумма калиброванных значений релевантности всех результатов поиска. CG в позиции p ранжированного списка определяется так:

$$CG_p \stackrel{\text{def}}{=} \sum_{i=1}^p rel_i,$$

где rel_i – относительная релевантность результата в позиции i . В общем случае относительная релевантность отражает релевантность документа запросу по дискретной шкале оценок, которые могут выражаться числами, буквами или словесными описаниями («не релевантен», «слабо релевантен», «релевантен», «сильно релевантен»). Для использования в приведенной выше формуле значения rel_i должны быть числовыми, например от 0 (документ в позиции i совершенно не релевантен запросу) до 1 (документ в позиции i максимально релевантен запросу). Или же rel_i может принимать бинарные значения: 0, если документ не релевантен запросу, и 1, если релевантен. Отметим, что CG_p не зависит от позиции документов в ранжированном списке. Он лишь характеризует документы, ранжированные до позиции p включительно, как релевантные или нерелевантные запросу.

Идея дисконтированного накопительного выигрыша основана на двух предположениях:

- 1) сильно релевантные документы полезнее, если находятся в начале списка результатов;
- 2) сильно релевантные документы полезнее слабо релевантных, которые, в свою очередь, полезнее нерелевантных.

Для данного результата поиска DCG в позиции p ранжированного списка часто определяется как

$$\text{DCG}_p \stackrel{\text{def}}{=} \sum_{i=1}^p \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^p \frac{rel_i}{\log_2(i+1)}.$$

Альтернативное определение DCG, нередко применяемое в индустрии и в конкурсах на тему науки о данных, например Kaggle, придает больший вес извлечению релевантных документов:

$$\text{DCG}_p \stackrel{\text{def}}{=} \sum_{i=1}^p \frac{2^{rel_i} - 1}{\log_2(i+1)}.$$

Для заданного запроса **нормированный дисконтированный накопительный выигрыш** (nDCG) определяется как

$$\text{nDCG}_p \stackrel{\text{def}}{=} \frac{\text{DCG}_p}{\text{IDCG}_p},$$

где IDCG – идеальный дисконтированный накопительный выигрыш

$$\text{IDCG}_p \stackrel{\text{def}}{=} \sum_{i=1}^{|\text{REL}_p|} \frac{2^{rel_i} - 1}{\log_2(i+1)},$$

a REL_p представляет список документов корпуса, релевантных запросу, до позиции p включительно (отсортированный по релевантности). То есть REL_p – идеальное ранжирование вплоть до позиции p , которое алгоритм ранжирования (или модель) поисковой системы должен был бы вернуть в ответ на запрос. Значения nDCG обычно усредняются по всем запросам, чтобы получить меру качества алгоритма или модели ранжирования.

Рассмотрим пример. Пусть поисковая система возвращает список документов в ответ на запрос. Мы просим ранжировщика (человека) оценить релевантность каждого документа, т. е. назначить оценку от 0 до 3, где 0 означает «не релевантен», 3 – «сильно релевантен», а 1 и 2 – «где-то посерединке». Допустим, что документы были выданы в следующем порядке:

$$D_1, D_2, D_3, D_4, D_5.$$

Ранжировщик поставил такие оценки релевантности:

$$3, 1, 0, 3, 2.$$

Это означает, что релевантность документа D_1 равна 3, документа D_2 – 1, D_3 – 0 и т. д. Накопительный выигрыш этого результата поиска до позиции $p = 5$ равен

$$\text{CG}_5 = \sum_{i=1}^5 rel_i = 3 + 1 + 0 + 3 + 2 = 9.$$

Как видим, изменение порядка документов не влияет на накопительный выигрыш. Теперь вычислим дисконтированный накопительный выигрыш

с логарифмическим дисконтированием, идея которого в том, чтобы повысить значение, если сильно релевантные документы находятся близко к началу списка. Для нахождения DCG_5 вычислим значение выражения $rel_i / \log_2(i + 1)$ для всех i :

i	rel_i	$\log_2(i + 1)$	$\frac{rel_i}{\log_2(i + 1)}$
1	3	1.00	3.00
2	1	1.58	0.63
3	0	2.00	0.00
4	3	2.32	1.29
5	2	2.58	0.77

Следовательно, DCG_5 этого ранжирования равен $3.00 + 0.63 + 0.00 + 1.29 + 0.77 = 5.70$.

Если переставить местами позиции D_1 и D_2 , то значение DCG_5 уменьшится. Это объясняется тем, что теперь менее релевантный документ ранжирован выше, а более релевантный обесценен сильнее, потому что оказался в более низкой позиции.

Чтобы вычислить нормированный дисконтированный накопительный выигрыш, $nDCG_5$, мы должны сначала найти значение дисконтированного накопительного выигрыша идеального упорядочения, $IDCG_5$. Согласно оценкам релевантности, идеальным является упорядочение 3, 3, 2, 1, 0. Тогда $IDCG_5 = 3.00 + 1.89 + 1.00 + 0.43 + 0.0 = 6.32$. Наконец, $nDCG_5$ равно

$$nDCG_5 = \frac{DCG_5}{IDCG_5} = \frac{5.70}{6.32} = 0.90.$$

Чтобы найти $nDCG$ для набора тестовых запросов и соответствующих списков результатов запросов, мы усредняем значения $nDCG_p$ для отдельных запросов. Преимущество нормированного дисконтированного накопительного выигрыша перед другими мерами состоит в том, что значения $nDCG_p$ для разных p можно сравнивать. Это свойство полезно, когда количество p оценок релевантности, поставленных ранжировщиками разным запросам, различается.

Имея метрику качества, мы можем сравнивать модели в процедуре, которая называется настройкой гиперпараметров.

5.6. НАСТРОЙКА ГИПЕРПАРАМЕТРОВ

Гиперпараметры играют важную роль в процессе обучения модели. Некоторые гиперпараметры влияют на скорость обучения, но большинство контролирует два компромисса: между смещением и дисперсией и между точностью и полнотой.

Гиперпараметры не оптимизируются алгоритмом обучения. Аналитик данных «настраивает» их, экспериментируя с комбинациями значений.

У каждой модели машинного обучения и каждого алгоритма обучения есть свой набор гиперпараметров. Кроме того, гиперпараметры могут быть на каждом шаге конвейера машинного обучения: предобработки данных, выделения признаков, обучения модели и предсказания.

Например, в случае предобработки данных гиперпараметры могут определять, нужно ли использовать какой-нибудь метод подстановки отсутствующих значений и какой именно. На этапе конструирования признаков гиперпараметр может определять используемый метод выделения признаков. На этапе предсказания, если модель возвращает оценку, гиперпараметр может определять порог принятия решения для каждого класса.

Ниже рассмотрено несколько популярных методов **настройки гиперпараметров**.

5.6.1. Поиск на сетке

Поиск на сетке – самый простой метод настройки гиперпараметров. Он применяется, когда число гиперпараметров и множество их допустимых значений не очень велики.

Объясним на примере настройки двух числовых гиперпараметров. Идея состоит в том, чтобы дискретизировать множество значений каждого гиперпараметра, а затем оценить каждую пару дискретных значений, как показано на рис. 5.6.

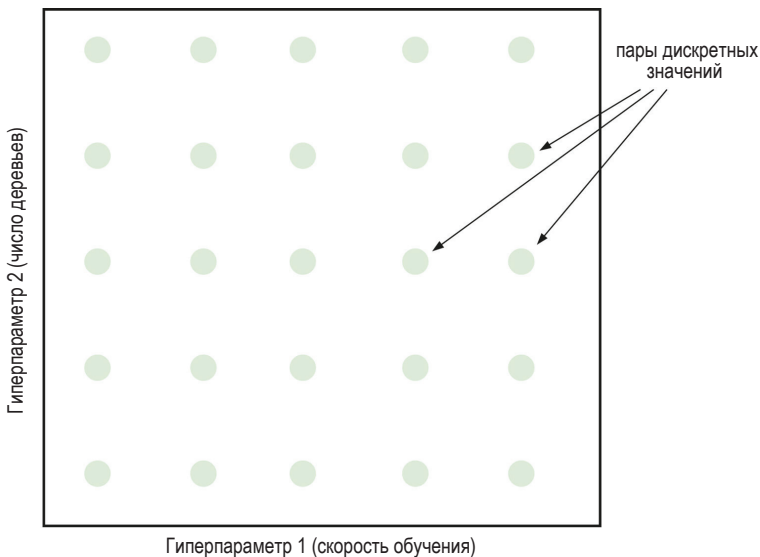


Рис. 5.6 ❖ Поиск на сетке для двух гиперпараметров: зеленые кружочки представляют пары значений гиперпараметров

Каждый раунд оценивания состоит из следующих шагов:

- 1) сконфигурировать конвейер, задав пару значений гиперпараметров;

- 2) применить конвейер к обучающим данным и обучить модель;
- 3) вычислить метрику качества модели на контрольных данных.

Затем для обучения окончательной модели выбирается пара гиперпараметров, для которой получилась наилучшая модель.

В приведенном ниже коде на Python используется поиск на сетке с перекрестной проверкой¹. Показано, как оптимизировать гиперпараметры простого двухэтапного конвейера `scikit-learn`, рассмотренного выше.

```

1 from sklearn.pipeline import Pipeline
2 from sklearn.svm import SVC
3 from sklearn.decomposition import PCA
4 from sklearn.model_selection import GridSearchCV
5
6 # Определить конвейер
7 pipe = Pipeline([('dim_reduction', PCA()), ('model_training', SVC())])
8
9 # Определить опробуемые значения гиперпараметров
10 param_grid = dict(dim_reduction__n_components=[2, 5, 10], \
11 model_training__C=[0.1, 10, 100])
12
13 grid_search = GridSearchCV(pipe, param_grid=param_grid)
14
15 # Сделать предсказание
16 pipe.predict(new_example)
```

В этом примере мы используем поиск на сетке для опробования значений [2, 5, 10] гиперпараметра PCA `n_components` и значений [0.1, 10, 100] гиперпараметра SVM `C`.

Для больших наборов данных опробование многих комбинаций гиперпараметров может занять много времени. Есть более эффективные методы, например случайный поиск, поиск с измельчением и байесовская оптимизация гиперпараметров.

5.6.2. Случайный поиск

Случайный поиск отличается от поиска на сетке тем, что не предоставляется дискретный набор значений для исследования каждого гиперпараметра. Вместо этого мы задаем для каждого гиперпараметра статистическое распределение, из которого случайным образом выбираются его значения. Кроме того, задается общее число комбинаций, как показано на рис. 5.7.

5.6.3. Поиск с измельчением

На практике часто применяется комбинация поиска на сетке и случайного поиска – **поиск с измельчением**. Сначала производится случайный грубый

¹ О перекрестной проверке мы поговорим в разделе 5.6.5.

поиск, чтобы найти перспективные области. Затем в этих областях производится поиск на мелкой сетке с целью найти наилучшие значения гиперпараметров (рис. 5.8).

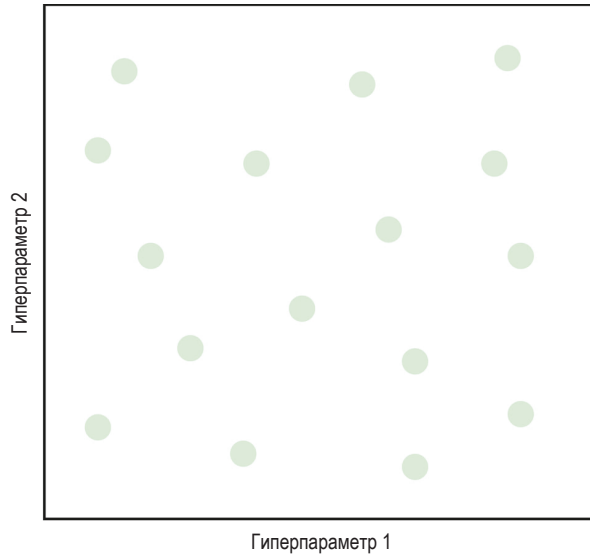


Рис. 5.7 ❖ Случайный поиск для двух гиперпараметров и 16 тестируемых пар

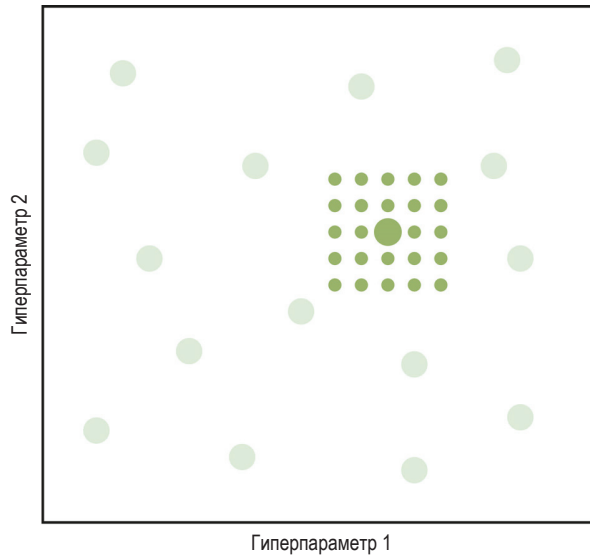


Рис. 5.8 ❖ Поиск с измельчением для двух гиперпараметров: 16 грубых случайных пар и один поиск на сетке в области, где случайный поиск нашел наилучшее значение

Сколько перспективных областей исследовать – одну или несколько, – зависит от располагаемого времени и вычислительных ресурсов.

5.6.4. Другие методы

Байесовские методы отличаются от случайного поиска и поиска на сетке тем, что используют прошлые результаты оценивания для решения о том, какие значения оценивать дальше. На практике это позволяет применить байесовские методы оптимизации гиперпараметров, чтобы найти наилучшие значения за меньшее время.

Существуют также градиентные методы, методы эволюционной оптимизации и другие алгоритмы настройки гиперпараметров. В большинстве современных библиотек машинного обучения есть реализации одного или нескольких методов подобного рода. Имеются также специальные библиотеки, которые позволяют настроить гиперпараметры практически любого алгоритма обучения, включая придуманный вами.

5.6.5. Перекрестная проверка

Поиск на сетке и другие рассмотренные выше методы настройки гиперпараметров применимы только тогда, когда имеется достаточно большой контрольный набор¹. Если такового нет, то для оценивания модели применяется **перекрестная проверка**. В самом деле, когда обучающих примеров мало, мы не можем позволить себе зарезервировать значительную их часть для тестового и контрольного наборов. Мы предпочли бы использовать больше данных для обучения модели. В таком случае следует разделить данные только на два набора: обучающий и тестовый. А затем использовать перекрестную проверку на обучающем наборе для имитации контрольного набора.

Перекрестная проверка работает следующим образом. Сначала мы фиксируем подлежащие оцениванию значения гиперпараметров. Затем делим обучающий набор на несколько поднаборов одинакового размера. Каждый поднабор называется группой. Обычно используется **пятигрупповая перекрестная проверка**, и мы случайным образом делим обучающие данные на пять групп: $\{F_1, F_2, \dots, F_5\}$. Каждая группа F_k , $k = 1, \dots, 5$, содержит 20 % обучающих данных. Затем мы обучаем пять моделей специальным образом. Для обучения первой модели, f_1 , используются все примеры из групп F_2, F_3, F_4 и F_5 в качестве обучающего набора и примеры из группы F_1 в качестве контрольного набора. Для обучения второй модели, f_2 , используются все примеры из групп F_1, F_3, F_4 и F_5 в качестве обучающего набора и примеры из группы F_2 в качестве контрольного. Так мы последовательно² обучаем модели f_k и вычисляем интересующую нас метрику на контрольном наборе. После этого все

¹ Достаточно большой значит содержащий по меньшей мере сотню примеров и такой, что каждый класс представлен по меньшей мере двадцатью примерами.

² Процесс перекрестной проверки проще объяснять как последовательный, хотя, конечно, все пять моделей от F_1 до F_5 можно строить параллельно.

пять значений метрики усредняются и результат считается окончательным значением. В общем случае при n -групповой перекрестной проверке модель f_n обучается на всех группах, кроме n -й.

Для нахождения наилучших значений гиперпараметров можно использовать поиск на сетке, случайный поиск или любой другой метод. После того как значения определены, мы обычно используем весь обучающий набор, чтобы обучить окончательную модель с лучшими значениями гиперпараметров, найденными в ходе перекрестной проверки. И наконец, полученная модель оценивается на тестовом наборе.

При поиске наилучших гиперпараметров может возникнуть искушение проверить все возможные значения, пусть это и нереалистично. Но помните, что время дорого, а лучшее – зачастую враг хорошего. Разверните в производственном режиме «достаточно хорошую» модель, а затем можете продолжить поиск идеальных значений гиперпараметров (пусть даже это займет несколько недель).

Теперь рассмотрим проблемы, возникающие при обучении поверхностной модели.

5.7. ОБУЧЕНИЕ ПОВЕРХНОСТНОЙ МОДЕЛИ

Поверхностные модели делают предсказания непосредственно по значениям входного вектора признаков. Самые популярные алгоритмы машинного обучения порождают поверхностные модели. Глубокими же моделями на сегодня являются только глубокие нейронные сети. Стратегию их обучения мы рассмотрим в разделе 6.1 следующей главы.

5.7.1. Стратегия обучения поверхностной модели

Типичная стратегия, применяемая в алгоритмах поверхностного обучения, выглядит следующим образом:

1. Определить метрику качества P .
2. Составить короткий список алгоритмов обучения.
3. Выбрать стратегию настройки гиперпараметров T .
4. Выбрать алгоритм обучения A .
5. Выбрать комбинацию H значений гиперпараметров алгоритма A с помощью стратегии T .
6. Используя обучающий набор, обучить модель M с помощью алгоритма A с гиперпараметрами H .
7. Используя контрольный набор, вычислить значение метрики P для модели M .
8. Принять решение:
 - а) если еще остались непротестированные значения гиперпараметров, выбрать другую комбинацию H значений гиперпараметров с помощью стратегии T и перейти к шагу 6;

b) в противном случае выбрать другой алгоритм обучения A и перейти к шагу 5 или перейти к шагу 9, если больше не осталось алгоритмов обучения.

9. Вернуть модель с максимальным значением метрики P .

Здесь на шаге 1 определяется метрика качества для задачи. В разделе 5.5 мы видели, что это математическая функция или подпрограмма, которая принимает на входе модель и набор данных и возвращает числовое значение, показывающее, насколько хорошо модель работает.

На шаге 2 мы выбираем алгоритмы-кандидаты и составляем из них короткий список (обычно два или три алгоритма). Для этого используется критерий отбора, рассмотренный в разделе 5.3.

На шаге 3 выбирается стратегия настройки гиперпараметров. Это последовательность действий, порождающая комбинации подлежащих тестированию значений гиперпараметров. Несколько таких стратегий мы обсудили в разделе 5.6.

5.7.2. Сохранение и восстановление модели

Обученную модель или конвейер необходимо сохранить в файле, чтобы потом развернуть в производственной среде и использовать для предсказания. Как модель, так и конвейер можно сериализовать. В Python для сериализации (сохранения) и десериализации (восстановления) объектов обычно применяется формат **Pickle**, а в R – формат RDS.

Ниже показано, как производятся сериализация и десериализация на Python.

```
1 import pickle
2 from sklearn.svm import SVC
3 from sklearn import datasets
4
5 # Подготовить данные
6 X, y = datasets.load_iris(return_X_y=True)
7
8 # Создать экземпляр модели
9 model = SVC()
10
11 # Обучить модель
12 model.fit(X, y)
13
14 # Сохранить модель в файле
15 pickle.dump(model, open("model_file.pkl", "wb"))
16
17 # Восстановить модель из файла
18 restored_model = pickle.load(open("model_file.pkl", "rb"))
19
20 # Сделать предсказание
21 prediction = restored_model.predict(new_example)
```

Аналогичный код на R выглядит так:

```

1 library("e1071")
2
3 # Подготовить данные
4 attach(iris)
5 X <- subset(iris, select=-Species)
6 y <- Species
7
8 # Обучить модель
9 model <- svm(X,y)
10
11 # Сохранить модель в файле
12 saveRDS(model, "./model_file.rds")
13
14 # Восстановить модель из файла
15 restored_model <- readRDS("./model_file.rds")
16
17 # Сделать предсказание
18 prediction <- predict(restored_model, new_example)

```

Теперь поговорим об особенностях процесса обучения модели, которые аналитики должны учитывать на практике, чтобы получить оптимальную модель.

5.8. КОМПРОМИСС МЕЖДУ СМЕЩЕНИЕМ И ДИСПЕРСИЕЙ

Разработка модели включает как поиск оптимального алгоритма, так и нахождение наилучших значений гиперпараметров. Гиперпараметры фактически контролируют два компромисса. Первый – между точностью и полнотой – мы уже обсудили. Второй – не менее важный – **компромисс между смещением и дисперсией**.

5.8.1. Недообучение

Говорят, что модель имеет **низкое смещение**, если она хорошо предсказывает метки на обучающем наборе. Если модель допускает слишком много ошибок на обучающих данных, то говорят, что она имеет **высокое смещение**, или что она **недообучена**. Есть несколько потенциальных причин недообученности:

- модель слишком проста для данных (например, линейные модели часто недообучены);
- признаки недостаточно информативны;
- модель слишком сильно регуляризирована (о регуляризации мы поговорим в следующем разделе).

Пример недообученной модели регрессии показан на рис. 5.9 (слева). Линия регрессии не повторяет изгибы кривой, которой, по всей видимости,

принадлежат данные. Модель слишком упрощает данные. Возможные решения проблемы недообучения таковы:

- попробовать более сложную модель;
- сконструировать признаки с большей **предсказательной способностью**;
- если возможно, добавить обучающие данные;
- уменьшить степень регуляризации.

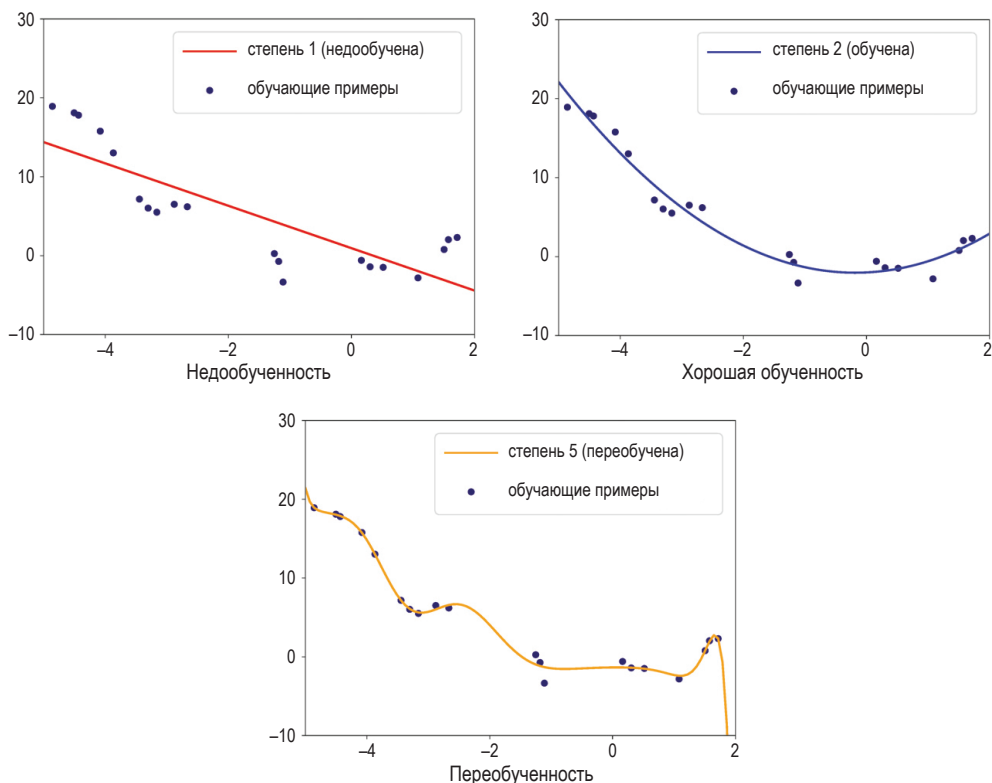


Рис. 5.9 ❖ Примеры недообученности (линейная модель), хорошей обученности (квадратичная модель) и переобученности (полиномиальная модель степени 5)

5.8.2. Переобучение

Переобученность – другая проблема, от которой может страдать модель. Переобученная модель обычно очень хорошо предсказывает метки обучающих данных, но плохо работает на зарезервированных данных.

Пример переобученной модели регрессии показан на рис. 5.9 (справа). Линия регрессии почти точно проходит через все обучающие примеры, но, скорее всего, модель будет допускать большие ошибки на новых данных, если мы рискнем использовать ее для предсказания.

В литературе встречается другое название переобученности: **высокая дисперсия**. Модель излишне чувствительна к малым флуктуациям обучающих данных. Если бы выборка обучающих данных была другой, то мы получили бы совершенно иную модель. Такие переобученные модели плохо ведут себя на зарезервированных данных, поскольку зарезервированные и обучающие примеры выбирались из набора данных независимо. Поэтому малые флуктуации в них, вероятно, будут различны.

К переобучению ведут несколько причин:

- модель слишком сложна для данных. Очень высокие решающие деревья и очень глубокие нейронные сети части бывают переобученными;
- слишком много признаков и слишком мало обучающих примеров;
- модель недостаточно регуляризирована.

Проблему переобучения можно решить несколькими способами:

- использовать более простую модель. Попробуйте линейную регрессию вместо полиномиальной либо SVM с линейным, а не **радиально-базисным** (RBF) ядром, либо нейронную сеть с меньшим числом слоев или блоков¹;
- уменьшить размерность примеров в наборе данных;
- если возможно, добавить обучающие данные;
- регуляризовать модель.

5.8.3. Компромисс

На практике попытка уменьшить дисперсию приводит к увеличению смещения, и наоборот. Иначе говоря, снижение переобученности ведет к недообученности, и наоборот. Это называется **компромиссом между смещением и дисперсией**: слишком усердно пытаясь построить модель, которая хорошо ведет себя на обучающих данных, мы в конечном итоге получаем модель, демонстрирующую плохое поведение на зарезервированных данных.

Есть много факторов, от которых зависит хорошее поведение модели на обучающих данных, но самый важный из них – сложность модели. Достаточно сложная модель обучится запоминать все обучающие примеры и их метки, а значит, не будет делать ошибок при предсказании меток обучающих данных. Однако модель, опирающаяся на запоминание, не сможет правильно предсказывать метки данных, которых ранее не видела. Поэтому ее дисперсия будет велика.

На рис. 5.10 показан типичный график средней ошибки предсказания на обучающих и зарезервированных данных с ростом сложности модели.

¹ Хотя уменьшение числа параметров модели часто рекомендуют, чтобы снизить степень переобучения и улучшить обобщаемость модели, феномен **двойного снижения** в глубокой сети иногда доказывает обратное. Этот феномен наблюдался в разных архитектурах, включая СНС и **трансформеры**: качество на контрольном наборе сначала улучшается, потом ухудшается, а потом снова улучшается с ростом размера модели. По состоянию на июль 2020 г. не вполне понятно, почему это происходит.

Мы хотели бы оказаться в «зоне решений» голубого цвета, где и дисперсия, и смещение малы. В пределах этой зоны мы можем точно настроить гиперпараметры, так чтобы достичь необходимого соотношения точности и полноты, или оптимизировать какую-то другую метрику качества, более соответствующую задаче.

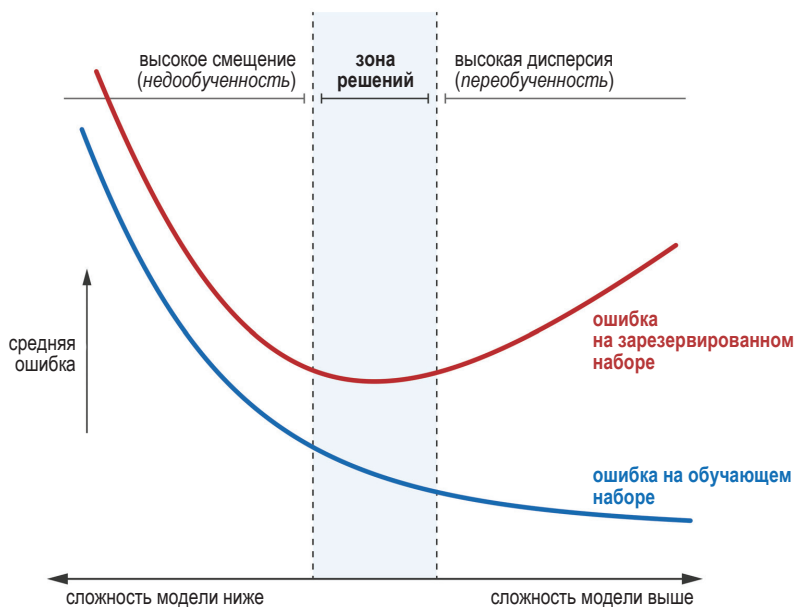


Рис. 5.10 ❖ Компромисс между смещением и дисперсией

Чтобы попасть в эту зону, есть два пути:

- двигаться слева направо, увеличивая сложность модели и, как следствие, уменьшая смещение;
- двигаться справа налево, регуляризируя модель, чтобы сделать ее проще и тем самым уменьшить дисперсию (о регуляризации мы будем говорить в следующем разделе).

Работая с поверхностными моделями типа линейной регрессии, мы можем увеличить сложность, перейдя к полиномиальной регрессии более высокого порядка. Аналогично можно увеличить глубину решающего дерева или использовать вместо линейного полиномиальное либо RBF-ядро в методе опорных векторов. Ансамблевые алгоритмы обучения, основанные на идее бустинга, позволяют уменьшить смещение путем комбинирования нескольких (возможно, сотен) «слабых» моделей с высоким смещением.

При работе с нейронными сетями сложность модели можно повысить, увеличив размер сети: количество блоков в слое и количество слоев. Обучение нейронной сети в течение более длительного времени (т. е. увеличение числа эпох) обычно тоже уменьшает смещение. Достоинством нейронных сетей в том, что касается компромисса между смещением и дисперсией, является то, что, немного увеличив размер сети, мы наблюдаем небольшое

уменьшение смещения. Большинство популярных поверхностных моделей и соответствующих алгоритмов обучения не обладают такой гибкостью.

Если в результате увеличения сложности модели вы попадете в правую часть графика на рис. 5.10, то следует уменьшить дисперсию модели. Самый популярный способ сделать это – регуляризация.

5.9. РЕГУЛЯРИЗАЦИЯ

Регуляризация – общий термин, обозначающий методы, которые вынуждают алгоритм обучить менее сложную модель. На практике это ведет к повышению смещения, но значительно уменьшает дисперсию.

Чаще всего применяется **L1-** или **L2-регуляризация**. Идея очень проста. Чтобы создать регуляризованную модель, мы модифицируем целевую функцию – то выражение, которое оптимизируется алгоритмом обучения модели. Регуляризация добавляет штрафующий член, который тем больше, чем сложнее модель.

Для простоты проиллюстрируем регуляризацию на примере линейной регрессии, но тот же принцип применим к широкому кругу моделей.

Пусть \mathbf{x} – двумерный вектор признаков $[x^{(1)}, x^{(2)}]$. Напомним целевую функцию линейной регрессии:

$$\min_{w^{(1)}, w^{(2)}, b} \left[\frac{1}{N} \times \sum_{i=1}^N (f_i - y_i)^2 \right]. \quad (5.3)$$

Здесь $f_i \stackrel{\text{def}}{=} f(\mathbf{x}_i)$, а f – уравнение прямой регрессии, имеющее вид $f = w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + b$. Алгоритм обучения выводит значения параметров $w^{(1)}$, $w^{(2)}$ и b из обучающих данных, минимизируя целевую функцию. Модель считается менее сложной, если некоторые параметры $w^{(i)}$ равны нулю или близки к нему.

5.9.1. L1- и L2-регуляризации

L1-регуляризованная целевая функция (5.3) имеет вид:

$$\min_{w^{(1)}, w^{(2)}, b} \left[C \times (|w^{(1)}| + |w^{(2)}|) + \frac{1}{N} \times \sum_{i=1}^N (f_i - y_i)^2 \right], \quad (5.4)$$

где C – **гиперпараметр**, управляющий важностью регуляризации. Если положить C равным нулю, то мы вернемся к стандартной нерегуляризованной модели линейной регрессии. С другой стороны, если взять большое C , то алгоритм обучения будет стараться сделать большинство параметров $w^{(i)}$ равными или близкими к нулю, чтобы доставить минимум целевой функции. Модель станет слишком простой, что может привести к недообучению. Роль аналитика данных – найти такое значение гиперпараметра C , при котором смещение увеличивается не слишком сильно, но дисперсия уменьшается до приемлемого в задаче уровня.

L2-регуляризованная целевая функция в нашей двумерной постановке выглядит так:

$$\min_{w^{(1)}, w^{(2)}, b} \left[C \times ((w^{(1)}) + (w^{(2)})^2) + \frac{1}{N} \times \sum_{i=1}^N (f_i - y_i)^2 \right]. \quad (5.5)$$

На практике L1-регуляризация порождает **разреженную модель** в предположении, что значение гиперпараметра C достаточно велико. В подобной модели большинство параметров в точности равно нулю. Таким образом, L1-регуляризация неявно производит **отбор признаков** (см. предыдущую главу), решая, какие признаки важны для предсказания, а какие нет. Это свойство L1-регуляризации полезно, когда мы хотим улучшить **объяснимость** модели. Но если наша цель – максимизировать качество модели на зарезервированных данных, то обычно L2-регуляризация дает лучший результат.

В литературе также встречаются названия **lasso** для L1-регуляризации и **гребневая регуляризация** или **уменьшение весов** для L2-регуляризации.

5.9.2. Другие формы регуляризации

L1 и L2 можно объединить и получить **регуляризацию эластичной сети**.

L1- и L2-регуляризации широко используются не только в применении к линейным моделям, но и к нейронным сетям и многим другим типам моделей, которые напрямую минимизируют целевую функцию.

Нейронные сети также могут получить выигрыш от двух других методов регуляризации: **прореживания** и **пакетной нормировки**. Существуют также нематематические методы, дающие эффект регуляризации: **приращение данных** и **ранняя остановка**. Мы поговорим о них более подробно в следующей главе, когда будем обсуждать обучение нейронных сетей.

5.10. РЕЗЮМЕ

Прежде чем приступить к работе над моделью, необходимо выполнить несколько проверок и принять несколько решений. Сначала убедитесь, что данные соответствуют схеме, определенной файлом схемы. Затем определите достижимый уровень качества и выберите метрику качества. В идеале она должна представлять качество модели одним числом. Далее важно установить ориентир, с которым будет сравниваться качество ваших моделей машинного обучения. Наконец, разбейте данные на три набора: обучающий, контрольный и тестовый.

В большинстве современных реализаций алгоритмов обучения классификаторов требуется, чтобы метки обучающих примеров были числовыми, поэтому мы, как правило, должны преобразовать метки в числовые векторы. Есть два популярных способа сделать это: унитарное кодирование (для бинарных и многоклассовых задач) и мешок слов (для многозначных задач).

Чтобы выбрать алгоритм машинного обучения, оптимальный для решения задачи, задайте себе следующие вопросы.

- Обязательно ли, чтобы предсказания модели можно было объяснить технически не подготовленной аудитории? Если да, то лучше использовать менее точные, но объяснимые алгоритмы, например kNN, линейная регрессия и решающие деревья.
- Можно ли загрузить весь набор данных в оперативную память ноутбука или сервера? Если нет, то лучше предпочесть инкрементные алгоритмы обучения.
- Сколько обучающих примеров имеется в наборе данных и сколько признаков имеется в каждом примере? Некоторые алгоритмы, включая те, что применяются для обучения нейронных сетей и случайных лесов, могут обрабатывать огромное число примеров и миллионы признаков. Емкость других скромнее.
- Являются ли данные линейно разделимыми, т. е. можно ли их смоделировать с помощью линейной модели? Если да, то имеет смысл выбрать SVM с линейным ядром, линейную или логистическую регрессию. В противном случае лучше остановиться на глубоких нейронных сетях или ансамблевых моделях.
- Сколько времени отведено алгоритму для обучения модели? Известно, что нейронные сети обучаются медленно. Простые алгоритмы, например линейная и логистическая регрессия или решающие деревья, гораздо быстрее.
- Насколько быстро модель должна делать предсказания в производственной среде? Модели типа SVM, линейной и логистической регрессии, а также не очень глубокие сети прямого распространения делают предсказания очень быстро. Для получения предсказаний от глубоких и рекуррентных нейронных сетей, а также моделей на основе градиентного бустинга требуется больше времени.

Если вы не хотите выбирать алгоритм наугад, то рекомендуется применить выборочную проверку к нескольким алгоритмам, а затем протестировать их на контрольном наборе, рассматривая выбор алгоритма как гиперпараметр.

Типичный способ узнать, насколько модель хороша, заключается в том, чтобы вычислить метрику качества на зарезервированных данных. Существуют метрики качества специально для моделей классификации и регрессии, а также для моделей ранжирования.

Значения гиперпараметров контролируют два компромисса: между точностью и полнотой и между смещением и дисперсией. Изменяя сложность модели, мы можем достичь так называемой «зоны решений», в которой и смещение, и дисперсия модели относительно малы. Решение, оптимизирующее метрику качества, обычно находится в этой зоне.

Регуляризация – общий термин, обозначающий методы, которые вынуждают алгоритм обучить менее сложную модель. На практике это ведет к повышению смещения, но значительно уменьшает дисперсию. Чаще всего применяется L1- или L2-регуляризация. Нейронные сети также могут получить выигрыш от двух других методов регуляризации: прореживания и пакетной нормировки.

Большинство современных пакетов и каркасов машинного обучения поддерживают понятие конвейера. Конвейер – это последовательность преобразований, которым подвергаются обучающие данные, прежде чем станут моделью. На каждой стадии конвейера к полученным на входе данным применяется некоторое преобразование. Каждая стадия конвейера, кроме первой, получает выход от предыдущей стадии. На вход первой стадии подается обучающий набор данных. Конвейер можно сохранить в файле по аналогии с сохранением модели. Его можно развернуть в производственной среде и использовать для генерирования предсказаний.

Гиперпараметры не оптимизируются самим алгоритмом обучения. Аналитик данных должен «настроить» их, экспериментируя с комбинациями значений. Поиск на сетке – простейший и самый популярный метод настройки гиперпараметров. Идея состоит в том, чтобы дискретизировать множество значений гиперпараметров, а затем попробовать все комбинации значений, для чего 1) обучить модель для каждой комбинации гиперпараметров и 2) вычислить метрики качества обученных моделей на контрольном наборе.

Достойный контрольный набор содержит по меньшей мере сотню примеров, а каждый класс в нем представлен по меньшей мере двадцатью примерами. Если получить такой набор невозможно, то следует прибегнуть к перекрестной проверке.

Глава 6

Обучение модели с учителем (часть 2)

Во второй части разговора об обучении моделей с учителем мы рассмотрим следующие темы: обучение глубоких моделей, штабелирование моделей, обработка несбалансированных наборов данных, сдвиг распределения, калибровка модели, поиск неполадок и анализ ошибок и другие передовые практики.

По сравнению с поверхностными моделями стратегия обучения глубоких нейронных сетей имеет больше деталей. С другой стороны, она лучше обоснована теоретически и более пригодна для автоматизации.

6.1. СТРАТЕГИЯ ОБУЧЕНИЯ ГЛУБОКИХ МОДЕЛЕЙ

Обучение модели начинается с составления короткого списка сетевых архитектур, или **топологий сети**. Если вы работаете с изображениями и хотите построить модель с нуля, то по умолчанию стоит выбрать **сверточную нейронную сеть** (СНС) хотя бы с одним **сверточным слоем**, за которым следует **слой max-пулинга** и один **полносвязный слой**.

При работе с текстом и другими последовательностями данных, например временными рядами, имеется выбор между СНС, **управляемой рекуррентной нейронной сетью** (например, долгой краткосрочной памятью (LSTM) или **управляемыми рекуррентными блоками** (GRU)) или **трансформером**.

Вместо того чтобы обучать модель с нуля, можно начать с **предобученной модели**. Такие компании, как Google и Microsoft, обучили очень глубокие нейронные сети с архитектурами, оптимизированными под задачи обработки изображений или естественного языка.

Из самых известных предобученных моделей для обработки изображений упомянем **VGG16** и **VGG19** (на основе архитектуры Visual Geometry Group, или **VGG**), **InceptionV3** (на основе архитектуры **GoogLeNet**) и **ResNet50** (на основе архитектуры **остаточной сети**).

Для обработки текстов на естественном языке имеются такие модели, как Bi-directional Encoder Representations from Transformer – **BERT** (на основе архитектуры трансформеров) и Embeddings from Language Models – **ELMo**

(на основе архитектуры **двунаправленной LSTM**). Их использование часто улучшает качество модели по сравнению с обучением с нуля.

Преимущество использования предобученных моделей в том, что они были обучены на гигантских объемах данных, доступных их создателям, но, скорее всего, недоступных вам. Даже если ваш набор данных меньше и не совсем похож на использованный для предобучения, параметры, которым обучилась предобученная модель, все равно могут оказаться полезны.

Использовать предобученные модели можно двумя способами:

- 1) взять обученные параметры в качестве начальных для своей модели;
- 2) использовать модель как экстрактор признаков для своей модели.

Если будете использовать предобученную модель первым способом, то получите дополнительную гибкость. Недостаток в том, что итогом обучения может оказаться очень глубокая нейронная сеть, а это потребует значительных вычислительных ресурсов. Во втором случае вы «замораживаете» параметры предобученной модели и обучаете только параметры добавленных слоев.

6.1.1. Стратегия обучения нейронной сети

Использование существующей модели для создания новой называется **переносом обучения**. Мы еще вернемся к этой теме в разделе 6.1.10. А пока предположим, что модель строится с нуля на основе выбранной нами архитектуры. Типичная стратегия построения нейронной сети выглядит следующим образом.

1. Определить метрику качества P .
2. Определить функцию стоимости C .
3. Выбрать стратегию инициализации параметров W .
4. Выбрать алгоритм оптимизации функции стоимости A .
5. Выбрать стратегию настройки гиперпараметров T .
6. Выбрать комбинацию N значений гиперпараметров, воспользовавшись стратегией настройки T .
7. Обучить модель M с помощью алгоритма A , параметризованного гиперпараметрами N , с целью оптимизации функции стоимости C .
8. Если остались непротестированные значения гиперпараметров, то выбрать еще одну комбинацию N значений гиперпараметров, пользуясь стратегией T , и перейти к шагу 7.
9. Вернуть модель, для которой метрика P наилучшая.

Теперь обсудим некоторые шаги этой стратегии подробнее.

6.1.2. Метрика качества и функция стоимости

Шаг 1 похож на шаг 1 стратегии обучения поверхностной модели (раздел 5.7): мы определяем метрику, которая позволила бы сравнить качество двух моделей на зарезервированных данных, и выбираем лучшую из двух. Примером метрики качества может служить **F-мера** или **каппа Коэна**.

На шаге 2 мы определяем, что именно будет оптимизировать алгоритм во время обучения модели. Если нейронная сеть является моделью регрессии,

то чаще всего в качестве **функции стоимости** выбирается **среднеквадратическая ошибка** (СКО), определенная в формуле (5.1) из предыдущей главы. Напомним ее:

$$\text{MSE}(f) \stackrel{\text{def}}{=} \frac{1}{N} \times \sum_{i=1 \dots N}^N (f(\mathbf{x}_i) - y_i)^2.$$

Для классификации в роли функции стоимости обычно выступает **категориальная перекрестная энтропия** (в многоклассовом случае) или **бинарная перекрестная энтропия** (в бинарном и многозначном случае).

Напомним, что при обучении нейронной сети для **многоклассовой классификации** метки должны быть представлены в **унитарной кодировке**. Пусть C – число классов в нашей задаче классификации. Обозначим \mathbf{y}_i – метку примера i в унитарной кодировке, где $i = 1, \dots, N$. Обозначим $y_{i,j}$ значение в позиции j (где $j = 1, \dots, C$) примера i . Потеря категориальной перекрестной энтропии на классификации примера i определяется как

$$\text{CCE}_i \stackrel{\text{def}}{=} - \sum_{j=1}^C [y_{i,j} \times \log_2(\hat{y}_{i,j})],$$

где $\hat{\mathbf{y}}_i$ – C -мерный вектор предсказания, выданный нейронной сетью для входа \mathbf{x}_i . Функция стоимости обычно определяется как сумма потерь на отдельных примерах:

$$\text{CCE} \stackrel{\text{def}}{=} \sum_{i=1}^N \text{CCE}_i.$$

При **бинарной классификации** выходом нейронной сети для входного вектора признаков \mathbf{x}_i является единственное значение \hat{y}_i , а меткой примера – единственное значение y_i , как в логистической регрессии. Потеря бинарной перекрестной энтропии на классификации примера i определяется как

$$\text{BCE}_i \stackrel{\text{def}}{=} -y_i \times \log_2(\hat{y}_i) - (1 - y_i) \times \log_2(1 - \hat{y}_i).$$

Аналогично функция стоимости для классификации обучающего набора обычно определяется как сумма потерь на отдельных примерах:

$$\text{BCE} \stackrel{\text{def}}{=} \sum_{i=1}^N \text{BCE}_i.$$

Бинарная перекрестная энтропия используется также при многозначной классификации. Теперь метками являются C -мерные векторы **мешков слов** \mathbf{y}_i , а предсказаниями – C -мерные векторы $\hat{\mathbf{y}}_i$, элементы которых $\hat{y}_{i,j}$ по каждому измерению j находятся в диапазоне от 0 до 1. Потеря на предсказании одной метки $\hat{\mathbf{y}}_i$ определяется как

$$\text{BCEM}_i \stackrel{\text{def}}{=} \sum_{j=1}^C [-y_{i,j} \times \log_2(\hat{y}_{i,j}) - (1 - y_{i,j}) \times \log_2(1 - \hat{y}_{i,j})].$$

Функция стоимости для классификации всего обучающего набора обычно определяется как сумма потерь на отдельных примерах

$$\text{BCEM} \stackrel{\text{def}}{=} \sum_{i=1}^N \text{BCEM}_i.$$

Заметим, что выходные слои при многоклассовой и многозначной классификациях различаются. В случае многоклассовой классификации используется один блок **softmax**. Он порождает C -мерный вектор с элементами в диапазоне $(0, 1)$, сумма которых равна 1. В случае многозначной классификации выходной слой содержит C логистических блоков, значения которых принадлежат диапазону $(0, 1)$, но сумма лежит в диапазоне $(0, C)$.

Выход нейронной сети

Любознательному читателю может быть интересно понять логику, стоящую за выбором конкретной функции потерь. Ниже мы математически опишем выход нейронной сети.

В случае регрессии выходной слой содержит всего один блок. Если на выходе может быть любое число от минус до плюс бесконечности, то выходной блок не будет содержать нелинейности.

С другой стороны, если нейронная сеть должна предсказывать положительное число, то можно использовать нелинейность типа **ReLU** (блок линейной ректификации). Обозначим z_i выходное значение выходного блока до применения нелинейности для входного примера i . Тогда после применения нелинейности ReLU получится значение $\max(0, z_i)$.

В случае бинарной классификации выходной слой содержит только один логистический блок. Обозначим z_i выходное значение выходного блока до применения нелинейности для входного примера i . Тогда выход \hat{y}_i после применения логистической нелинейности равен

$$\hat{y}_i \stackrel{\text{def}}{=} \frac{1}{1 + e^{-z_i}},$$

где e – **число Эйлера**, основание натуральных логарифмов.

Модели бинарной и многозначной классификаций определяются аналогично. Единственное отличие в том, что при многозначной классификации выходной слой содержит C логистических блоков, по одному на каждый класс. Если обозначить $\hat{y}_{i,j}$ выход логистического блока после применения нелинейности для класса j на входном примере i , то сумма $\hat{y}_{i,j}$ для всех $j = 1, \dots, C$ будет находиться между 0 и C .

При многоклассовой классификации выходной слой также порождает C выходов. Однако в этом случае выход каждого блока выходного слоя контролируется функцией **softmax**. Обозначим $\hat{z}_{i,j}$ выход выходного блока j для входного примера i до применения нелинейности. Тогда выход $\hat{y}_{i,j}$ после применения нелинейности равен

$$\hat{y}_{i,j} \stackrel{\text{def}}{=} \frac{e^{z_{i,j}}}{\sum_{k=1}^C e^{z_{i,k}}}.$$

Сумма $\hat{y}_{i,j}$ для всех $j = 1, \dots, C$, равна 1.

6.1.3. Стратегии инициализации параметров

На шаге 3 выбирается **стратегия инициализации параметров**. До начала обучения значения параметров во всех блоках неизвестны. Их нужно как-то инициализировать. Алгоритмы обучения нейронных сетей, в частности **градиентный спуск** и его стохастические варианты, которые мы рассмотрим чуть ниже, по природе своей итеративные, поэтому аналитик должен задать начальную точку, с которой начинаются итерации. Эта инициализация может повлиять на свойства обученной модели. Чаще всего выбирается одна из следующих стратегий:

- **все единицы** – все параметры инициализируются значением 1;
- **все нули** – все параметры инициализируются значением 0;
- **случайная нормальная** – параметры инициализируются значениями, выбранными из **нормального распределения** обычно со средним 0 и стандартным отклонением 0.05;
- **случайная равномерная** – параметры инициализируются значениями, выбранными из **равномерного распределения** на отрезке $[-0.05, 0.05]$;
- **нормальная Ксавье** – параметры инициализируются значениями, выбранными из усеченного нормального распределения с центром 0 и стандартным отклонением $\sqrt{2/(\text{in} + \text{out})}$, где in – число блоков предыдущего слоя, с которыми связан текущий блок (чьи параметры мы инициализируем), а out – число блоков следующего слоя, с которыми связан текущий блок;
- **равномерная Ксавье** – параметры инициализируются значениями, выбранными из равномерного распределения на отрезке $[-\text{limit}, \text{limit}]$, где limit равно $\sqrt{6/(\text{in} + \text{out})}$, а in и out определены, как для нормальной инициализации Ксавье выше.

Существуют и другие стратегии инициализации. Библиотеки обучения нейронных сетей, например TensorFlow, Keras или PyTorch, предоставляют инициализаторы параметров и рекомендуют режим по умолчанию.

Член смещения обычно инициализируется нулем.

Хотя мы знаем, что инициализация параметров влияет на свойства модели, предсказать, какая стратегия даст лучший результат для конкретной задачи, невозможно. Чаще всего применяются случайный инициализатор и инициализаторы Ксавье. Рекомендуем начинать эксперименты с этих двух стратегий.

6.1.4. Алгоритмы оптимизации

На шаге 4 мы выбираем алгоритм оптимизации функции стоимости. Если функция стоимости дифференцируемая (а таковыми являются все рассмот-

ренные выше функции стоимости), то для оптимизации чаще всего применяются алгоритмы **градиентного спуска** и **стохастического градиентного спуска**.

Градиентный спуск – это итеративный алгоритм оптимизации для нахождения **локального минимума** любой дифференцируемой функции. Говорят, что $f(x)$ имеет **локальный минимум** в точке $x = c$, если $f(x) \geq f(c)$ для любого x в некотором **открытом интервале**, окружающем $x = c$. **Интервалом** называется множество вещественных чисел такое, что любое число, расположенное между двумя числами, принадлежащими этому множеству, также принадлежит ему. Открытый интервал не включает концевых точек и обозначается круглыми скобками. Например, $(0, 1)$ означает «все числа, большие 0 и меньшие 1». Минимальное значение среди всех локальных минимумов называется **глобальным минимумом**. Разница между локальным и глобальным минимумами функции показана на рис. 6.1.

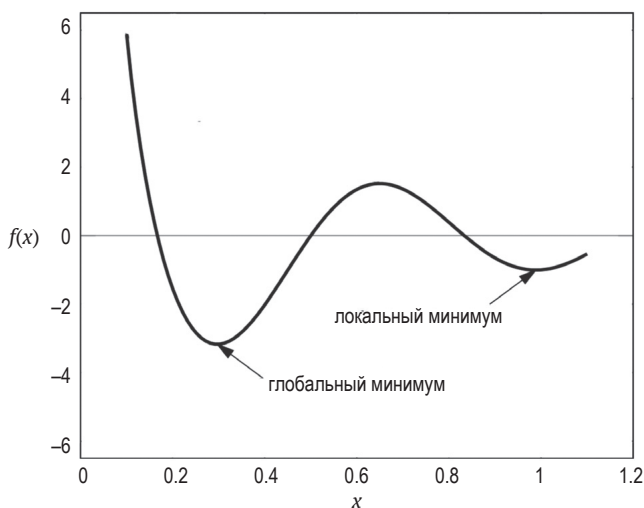


Рис. 6.1 ❖ Локальный и глобальный минимумы функции

Функции и оптимизация

Для любознательных читателей объясним, что такое математическая функция и оптимизация функций. Если вам интересен только механизм обучения нейронных сетей, можете пропустить эту часть.

Функцией называется отношение, при котором каждому элементу x множества X , называемого **областью определения** функции, ставится в соответствие единственный элемент y другого множества Y , называемого **областью значений** функции. Обычно функция имеет имя. Если функция называется f , то это отношение обозначается $y = f(x)$ и читается « y равно f от x ». Элемент x называется аргументом, или входом функции, а y – значением, или выходом функции. Символ, используемый для представления входа, называется переменной. Часто можно услышать выражение « f является функцией от переменной x ».

Производной f' функции f называется функция, которая описывает, насколько быстро f возрастает или убывает. Если производная постоянна, например равна 5 или -3 , то функция возрастает или убывает с постоянной скоростью в любой точке своей области определения. Если производная f' не постоянна, то f может изменяться с разной скоростью в разных точках области определения. Если производная f' в некоторой точке x положительна, то f возрастает в этой точке. Если же производная f' отрицательна, то функция в этой точке убывает. Если производная в точке x равна нулю, значит, функция в этой точке не возрастает и не убывает, т. е. касательная к ней горизонтальна.

Процесс нахождения производной называется **дифференцированием**. Производные простых функций известны. Например, если $f(x) = x^2$, то $f'(x) = 2x$; если $f(x) = 2x$, то $f'(x) = 2$; если $f(x) = 2$, то $f'(x) = 0$. Производная любой постоянной функции $f(x) = c$ равна нулю.

Если функция, которую мы хотим продифференцировать, не простая, то ее производную можно найти по **правилу дифференцирования сложной функции**. Именно, если $F(x) = f(g(x))$, где f и g – некоторые функции, то $F'(x) = f'(g(x))g'(x)$. Например, если $F(x) = (5x + 1)^2$, то $g(x) = 5x + 1$ и $f(g(x)) = (g(x))^2$. Применяя правило дифференцирования сложной функции, найдем $F'(x) = 2(5x + 1)g'(x) = 2(5x + 1)5 = 50x + 10$.

Градиент – это обобщение производной на функции от нескольких аргументов или от одного аргумента в виде вектора либо другой сложной структуры. Градиентом функции является вектор **частных производных**. Чтобы найти частную производную функции, нужно считать, что изменяется только один аргумент, а остальные фиксированы.

Например, для функции $f([x^{(1)}, x^{(2)}]) = ax^{(1)} + bx^{(2)} + c$ частная производная по $x^{(1)}$, обозначаемая $\partial f / \partial x^{(1)}$, равна

$$\frac{\partial f}{\partial x^{(1)}} = a + 0 + 0 = a,$$

где a – производная функции $ax^{(1)}$. Два нуля – это производные $bx^{(2)}$ и c , потому что $x^{(2)}$ считается постоянным при вычислении частной производной по $x^{(1)}$, а производная любой постоянной равна нулю.

Аналогично частная производная f по $x^{(2)}$, $\partial f / \partial x^{(2)}$, равна

$$\frac{\partial f}{\partial x^{(2)}} = 0 + b + 0 = b.$$

Градиент функции f , обозначаемый ∇f , – это вектор $[\partial f / \partial x^{(1)}, \partial f / \partial x^{(2)}]$.

Правило дифференцирования сложной функции имеет место и для частных производных.

Чтобы найти локальный минимум функции методом **градиентного спуска**, мы начинаем со случайной точки в области определения функции. Затем делаем шаг в направлении, противоположном градиенту, на величину, пропорциональную градиенту (быть может, приближенному) функции в текущей точке.

Градиентный спуск в машинном обучении разбит на **эпохи**. В одной эпохе используется весь обучающий набор целиком для обновления каждого параметра. В первой эпохе мы инициализируем параметры нейронной сети, применяя одну из рассмотренных выше стратегий инициализации. Алгоритм **обратного распространения** вычисляет частные производные каждого параметра с помощью правила дифференцирования сложной функции¹. В каждой эпохе алгоритм градиентного спуска обновляет все параметры с использованием частных производных. **Скорость обучения** контролирует величину обновления. Процесс продолжается, пока не **сойдется**, т. е. пока значения параметров не перестанут существенно изменяться от эпохи к эпохе. После этого алгоритм останавливается.

Градиентный спуск чувствителен к выбору скорости обучения α . Выбрать подходящую для задачи скорость обучения нелегко. Если выбрать слишком большое значение, то алгоритм может вовсе не сойтись. С другой стороны, при слишком малых α обучение замедляется настолько, что никакой прогресс не заметен. На рис. 6.2 показана иллюстрация градиентного спуска для одного параметра нейронной сети и трех значений скорости обучения. Значение параметра на каждой итерации показано голубым кружочком. Число внутри кружочка – номер эпохи. Красной стрелкой показано направление градиента вдоль горизонтальной оси – направление в сторону от минимума. Зеленой стрелкой показано, как изменяется значение функции стоимости после каждой эпохи.

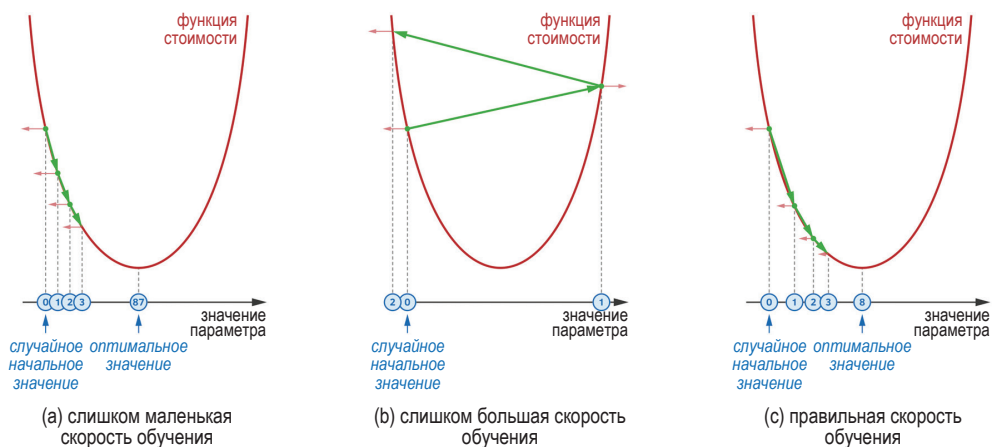


Рис. 6.2 ❖ Влияние скорости обучения на сходимость:

- (a) слишком маленькая, сходимость будет медленной; (b) слишком большая, сходимость отсутствует; (c) правильно выбранная скорость обучения

¹ Объяснение алгоритма обратного распространения выходит за рамки этой книги. Нужно только знать, что в любой современной библиотеке для обучения нейронных сетей его реализация имеется. Любознательный читатель может найти описание этого алгоритма в расширенной версии книги «The Hundred-Page Machine Learning Book» на сопроводительном сайте.

Таким образом, в каждой эпохе градиентный спуск сдвигает значение параметра в сторону минимума. Если скорость обучения слишком мала, то движение к минимуму очень медленное (рис. 6.2(a)). Если скорость обучения слишком велика, то значение параметра будет колебаться вокруг минимума (рис. 6.2(b)).

Градиентный спуск работает довольно медленно для больших данных, потому что для вычисления градиента каждого параметра в каждой эпохе используется весь набор данных. По счастью, было предложено несколько существенных улучшений этого алгоритма.

Стохастический градиентный спуск на мини-пакетах (мини-пакетный СГС) – вариант алгоритма градиентного спуска. В этом случае градиент аппроксимируется с помощью небольших подмножеств обучающих данных, называемых **мини-пакетами**. Это ускоряет вычисления. Размер мини-пакета – гиперпараметр, который можно настраивать. Рекомендуются степени 2 в диапазоне от 32 до нескольких сотен, например 32, 64, 128, 256 и т. д.

Задача выбора правильной скорости обучения α присутствует и в «прос-том» мини-пакетном СГС. В поздних эпохах обучение может застопориться. И тогда, вместо того чтобы найти локальный минимум, градиентный спуск будет колебаться вокруг него из-за слишком больших обновлений. Существует много **планов уменьшения скорости обучения**, которые позволяют изменять скорость по мере продвижения обучения, постепенно уменьшая ее. Такой подход ускоряет сходимость градиентного спуска (сеть обучается быстрее) и повышает качество модели. Ниже мы рассмотрим несколько популярных планов уменьшения скорости обучения.

6.1.5. Планы уменьшения скорости обучения

Идея состоит в постепенном уменьшении **скорости обучения** α при переходе от эпохи к эпохе. Следовательно, обновления параметров становятся мельче. Существует несколько методов, или планов управления α .

Временные планы уменьшения скорости обучения изменяют скорость обучения в зависимости от скорости в предыдущей эпохе. Математическая формула обновления скорости обучения в одном из популярных планов имеет вид

$$\alpha_n \leftarrow \frac{\alpha_{n-1}}{1 + d \times n},$$

где α_n – новое значение скорости обучения, α_{n-1} – скорость обучения в предыдущую эпоху $n - 1$, а d – **скорость уменьшения**, гиперпараметр. Например, если начальное значение скорости обучения $\alpha_0 = 0.3$, то значения скорости обучения в первые пять эпох показаны в таблице ниже.

Скорость обучения	Эпоха
0.15	1
0.10	2
0.08	3
0.06	4
0.05	5

Шаговые планы уменьшения скорости обучения изменяют скорость обучения некоторыми предопределенными шагами спада. Математическая формула обновления скорости обучения в одном из популярных планов такого рода имеет вид

$$\alpha_n \leftarrow \alpha_0 d^{\text{floor}\left(\frac{1+n}{r}\right)},$$

где α_n – скорость обучения в эпоху n , α_0 – начальное значение скорости обучения, d – темп уменьшения, который показывает, на сколько должна измениться скорость обучения на каждом шаге (0.5 соответствует уменьшению вдвое), а r – темп спада, определяющий длину шагов спада (10 соответствует падению через каждые 10 эпох). Функция floor в формуле выше возвращает 0, если значение ее аргумента меньше 1.

Экспоненциальные планы уменьшения скорости обучения похожи на шаговые. Но вместо шагов спада используется убывающая экспоненциальная функция. Математическая формула обновления скорости обучения в одном из популярных планов такого рода имеет вид

$$\alpha_n \leftarrow \alpha_0 e^{-d \times n},$$

где d – темп уменьшения, а e – **число Эйлера**.

Существует несколько усовершенствований **мини-пакетного ГГС**: Momentum, Root Mean Squared Propagation (RMSProp) и Adam. Эти алгоритмы обновляют скорость обучения автоматически, исходя из качества процесса обучения. Нам не нужно беспокоиться о выборе начальной скорости обучения, плана и темпа уменьшения и других гиперпараметров. Эти алгоритмы продемонстрировали хорошее качество на практике и часто используются вместо ручной настройки скорости обучения.

Momentum ускоряет мини-пакетный ГГС за счет выбора нужного направления градиентного спуска, в результате чего уменьшаются осцилляции. Для выбора направления поиска используется градиент не только в текущей эпохе, но и накопленный в прошлых эпохах. Momentum устраняет необходимость в ручной корректировке скорости обучения.

К более поздним достижениям в области алгоритмов оптимизации функции стоимости нейронной сети относятся **RMSProp** и **Adam**, причем последний (появившийся позже) более универсален. Рекомендуется начинать обучение модели с Adam. И только если качество модели не достигает приемлемого уровня, попробуйте другой алгоритм оптимизации.

6.1.6. Регуляризация

В нейронных сетях, помимо **L1- и L2-регуляризаций**, можно использовать специальные методы: прореживание, раннюю остановку и пакетную нормировку. Последняя технически не является методом регуляризации, но часто оказывает регуляризирующий эффект.

Идея **прореживания** очень проста. При каждом «прогоне» обучающего примера через сеть мы временно исключаем из вычисления некоторые

случайно выбранные блоки. Чем выше процент исключенных блоков, тем сильнее регуляризирующий эффект. Популярные библиотеки работы с нейронными сетями позволяют добавить слой прореживания между двумя соседними слоями или задать гиперпараметр прореживания для какого-то слоя. Этот гиперпараметр изменяется в диапазоне $[0, 1]$ и характеризует долю блоков, исключаемых из вычисления. Его значение следует подбирать экспериментально. Несмотря на простоту, прореживание обладает феноменальной гибкостью и регуляризирующим эффектом.

Метод ранней остановки обучает сеть путем сохранения предварительной модели после каждой эпохи. Сохраненные модели называются **контрольными точками**. Затем качество каждой контрольной точки оценивается на контрольном наборе. Во время градиентного спуска мы обнаруживаем, что стоимость уменьшается при увеличении количества эпох. После какой-то эпохи модель может начать проявлять склонность к переобучению, и ее качество на контрольных данных ухудшается. Вспомните иллюстрацию компромисса между смещением и дисперсией на рис. 5.10. Сохраняя версию модели после каждой эпохи, мы можем остановить обучение, заметив падение качества на контрольном наборе. Альтернативно можно продолжать процесс обучения на протяжении фиксированного количества эпох, а затем выбрать наилучшую контрольную точку. Некоторые специалисты по машинному обучению предпочитают именно такой метод. Другие стараются регуляризировать модель, применяя подходящие техники.

Пакетная нормировка (которую правильнее было бы называть пакетной стандартизацией) заключается в **стандартизации** выходов каждого слоя, перед тем как передать их на вход следующему слою. На практике пакетная нормировка приводит к более быстрому и устойчивому обучению, а также к определенному регуляризирующему эффекту. Поэтому применять ее всегда имеет смысл. В популярных библиотеках работы с нейронными сетями часто можно вставить слой пакетной нормировки между двумя соседними слоями.

Еще один метод регуляризации, применимый к любому алгоритму обучения, – **приращение данных**. Он часто используется для регуляризации моделей, работающих с изображениями. На практике приращение данных нередко улучшает качество модели.

6.1.7. Определение размера сети и настройка гиперпараметров

Шаг 5 стратегии обучения глубокой модели похож на шаг стратегии обучения поверхностной модели – выбрать стратегию настройки гиперпараметров T .

На шаге 6 мы выбираем комбинацию значений гиперпараметров с помощью стратегии T . Типичные параметры включают размер мини-пакета, значение скорости обучения (если используется стандартный мини-пакетный СГС) или алгоритм, который автоматически обновляет скорость обучения, например Adam. Мы также принимаем решение о начальном числе слоев и блоков в одном слое. Рекомендуется начать с чего-то разумного, что

позволило бы построить первую модель достаточно быстро. Например, два скрытых слоя и 128 блоков в слое можно считать неплохой отправной точкой.

Шаг 7 гласит: «Обучить модель M с помощью алгоритма A , параметризованного гиперпараметрами H , с целью оптимизации функции стоимости C ». В этом и состоит основное отличие от поверхностного обучения. При работе с алгоритмом или моделью поверхностного обучения мы можем только настроить некоторые встроенные гиперпараметры, а над архитектурой и сложностью модели у нас нет почти никакого контроля. В случае нейронных сетей все в наших руках, так что обучение модели становится скорее процессом, чем одним действием. Чтобы построить глубокую модель, мы начинаем с модели разумного размера, а затем действуем, как предлагает блок-схема на рис. 6.3.

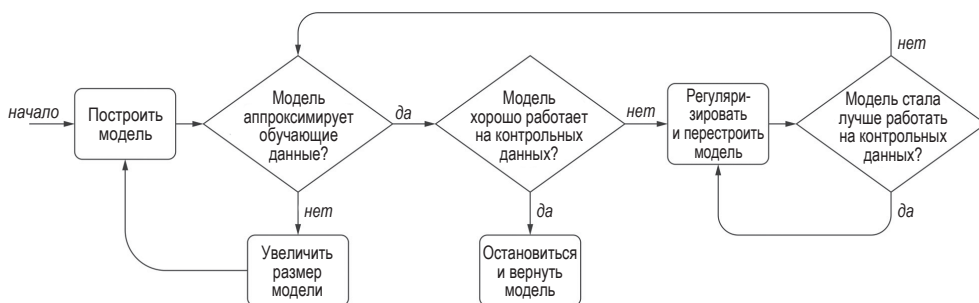


Рис. 6.3 ❖ Блок-схема обучения модели нейронной сети

Обратите внимание, что мы начинаем с какой-то модели, а затем увеличиваем ее размер, пока не добьемся хорошей аппроксимации обучающих данных. После этого мы оцениваем модель на контрольных данных. Если она хорошо работает с точки зрения выбранной метрики качества, то мы останавливаемся и возвращаем модель. В противном случае регуляризируем ее и переобучаем.

Как мы видели, регуляризовать нейронную сеть можно несколькими способами. Самый эффективный из них – **прореживание**, когда случайным образом из сети удаляются некоторые блоки, что делает ее проще и «глупее». Более простая модель должна лучше работать на зарезервированных данных, а это и есть наша цель.

Предположим, что после нескольких раундов регуляризации и переобучения модели мы не наблюдаем никакого улучшения качества на контрольных данных. Тогда нужно проверить, по-прежнему ли модель аппроксимирует обучающие данные. Если нет, увеличиваем размер модели путем увеличения размера отдельных слоев или добавления новых слоев. Продолжаем, пока модель не начнет снова аппроксимировать данные. После чего опять оцениваем ее на контрольных данных. Процесс продолжается до тех пор, пока не станет ясно, что как бы ни укрупнять модель, лучшего качества на контрольных данных не добиться. Тогда мы останавливаемся и возвращаем модель, если качество на контрольных данных удовлетворительно.

Если же мы не довольны достигнутым качеством, то можем выбрать другую комбинацию гиперпараметров на шаге 8 и построить другую модель. Испытание различных комбинаций гиперпараметров продолжается, пока не будут исчерпаны все возможные комбинации. Если качество лучшей из полученных моделей по-прежнему неудовлетворительно, следует попробовать другую архитектуру, добавить больше помеченных данных или прибегнуть к **переносу обучения**. Подробнее о переносе обучения мы будем говорить в разделе 6.1.10.

Свойства обученной нейронной сети сильно зависят от выбора значений гиперпараметров. Но прежде чем выбирать их значения, обучать модель и оценивать ее свойства на контрольных данных, необходимо решить, какие гиперпараметры достаточно важны, чтобы тратить на них время.

Понятно, что будь у нас бесконечно много времени и вычислительных ресурсов, мы настроили бы все гиперпараметры. Но на практике время ограничено, а ресурсы зачастую довольно скромны. Так какие же гиперпараметры настраивать?

Определенного ответа на этот вопрос нет, но некоторые соображения могут помочь при выборе состава гиперпараметров для конкретной модели:

- ваша модель чувствительнее к одним гиперпараметрам, чем к другим;
- выбирать часто приходится между использованием значения гиперпараметра по умолчанию и его изменением.

Библиотеки для обучения нейронных сетей часто уже предлагают значения некоторых гиперпараметров по умолчанию: версия стохастического градиентного спуска (чаще всего **Adam**), стратегия инициализации параметров (часто **случайная нормальная** или **случайная равномерная**), размер мини-пакета (обычно 32) и т. д. Эти значения основаны на практическом опыте. Библиотеки и модули с открытым исходным кодом зачастую являются плодом совместного труда многих ученых и инженеров. Эти талантливые и опытные люди определили «хорошие» значения по умолчанию для многих гиперпараметров в процессе работы с различными наборами данных и практическими задачами.

Если вы все-таки решили настраивать гиперпараметр, а не пользоваться значением по умолчанию, то разумно выбрать только те гиперпараметры, к которым модель наиболее чувствительна. В табл. 6.1 приведены¹ несколько гиперпараметров и приближенная чувствительность к ним нейронной сети.

6.1.8. Работа с несколькими входами

На практике инженерам по машинному обучению часто приходится работать с мультимодальными данными. Например, на вход могут быть поданы изображение и текст, а бинарный выход сообщает, верно ли, что текст описывает изображение.

Адаптировать алгоритмы **поверхностного обучения** к мультимодальным данным трудно. Например, можно было бы попытаться векторизовать все выходы, применив подходящий метод конструирования признаков. За-

¹ Взято из лекции «Troubleshooting Deep Neural Networks», прочитанной Джошем Тобиным и др. в январе 2019 года.

тем конкатенировать два вектора признаков, образовав более широкий вектор. Если изображение имеет признаки $[i^{(1)}, i^{(2)}, i^{(3)}]$, а текст – признаки $[t^{(1)}, t^{(2)}, t^{(3)}, t^{(4)}]$, то конкатенированный вектор будет равен $[i^{(1)}, i^{(2)}, i^{(3)}, t^{(1)}, t^{(2)}, t^{(3)}, t^{(4)}]$.

В случае нейронных сетей гибкости гораздо больше. Можно построить две **подсети**, по одной для входа каждого типа. Например, подсеть **СНС** будет читать изображение, а подсеть **РНС** – читать текст. Последним слоем в обеих подсетях является **погружение**: изображения в случае СНС и текста в случае РНС. Затем мы конкатенируем оба погружения и, наконец, добавляем слой классификации, например **softmax** или **логистическую сигмоиду**, применяемый к конкатенированным погружениям.

Таблица 6.1. Приблизительная чувствительность модели к некоторым гиперпараметрам

Гиперпараметр	Чувствительность
Скорость обучения	Высокая
План скоростей обучения	Высокая
Функция потерь	Высокая
Число блоков в одном слое	Высокая
Стратегия инициализации параметров	Средняя
Число слоев	Средняя
Свойства слоя	Средняя
Степень регуляризации	Средняя
Выбор оптимизатора	Низкая
Свойства оптимизатора	Низкая
Размер мини-пакета	Низкая
Выбор нелинейности	Низкая

Библиотеки обучения нейронных сетей предлагают простые в использовании инструменты, которые позволяют конкатенировать или усреднять слои, взятые из нескольких подсетей.

6.1.9. Работа с несколькими выходами

Иногда требуется предсказать несколько выходов по одному входу. Некоторые задачи с несколькими выходами можно эффективно преобразовать в задачу многозначной классификации. Так обстоит дело, когда метки имеют одну и ту же природу (как теги в социальных сетях) или когда можно создать фиктивные метки, являющиеся перечислением всех возможных комбинаций оригинальных меток.

Однако во многих случаях выходы мультимодальные, и их комбинации невозможно эффективно перечислить. Рассмотрим следующий пример: требуется построить модель, которая обнаруживает в изображении объект и возвращает его координаты. Кроме того, модель должна вернуть тег, описывающий объект, например «человек», «кошка» или «хомяк». Обучающими примерами будут векторы признаков, представляющие изображение и мет-

ку. Метку можно представить в виде вектора координат объекта и еще одного вектора, содержащего тег в **унитарной кодировке**.

Для этого мы можем создать подсеть, работающую как кодировщик. Она будет читать входное изображение, используя, например, один или несколько сверточных слоев. Последним слоем кодировщика будет погружение изображения. Затем добавим еще две подсети после слоя погружения: 1) одна принимает на входе вектор погружения и предсказывает координаты объекта, 2) другая принимает на входе вектор погружения и предсказывает тег.

В первой подсети последним слоем может быть **ReLU**, который хорошо подходит для предсказания положительных чисел, в частности координат. В этой подсети можно использовать функцию стоимости, равную среднеквадратической ошибке, C_1 . Вторая подсеть принимает на входе тот же вектор погружения и предсказывает вероятности каждого тега. Последним в ней может быть слой **softmax**, подходящий для многоклассовой классификации, а в качестве функции стоимости используется усредненное **отрицательное логарифмическое правдоподобие** C_2 (или **перекрестная энтропия**). Альтернативно координаты могли бы принадлежать диапазону $[0, 1]$ (в таком случае слой, предсказывающий координаты, будет иметь четыре **логистических сигмоидных** выхода и усреднять четыре функции стоимости **бинарной перекрестной энтропии**), а слой, предсказывающий теги, мог бы решать задачу **многозначной классификации** (тогда он тоже имел бы несколько сигмоидных выходов и усреднял несколько функций бинарной перекрестной энтропии, по одной для каждого тега).

Очевидно, мы хотели бы, чтобы и координаты, и теги предсказывались верно. Однако оптимизировать сразу обе функции стоимости невозможно. Пытаясь улучшить одно, мы рискуем ухудшить другое, и наоборот. А вот что можно сделать, так это добавить еще один гиперпараметр γ в диапазоне $(0, 1)$ и определить комбинированную функцию стоимости $\gamma \times C_1 + (1 - \gamma) \times C_2$. После этого γ можно будет настроить на контрольных данных, как любой другой гиперпараметр.

6.1.10. Перенос обучения

Напомним, что под **переносом обучения** понимается использование предобученной модели для построения новой модели. Предобученные модели обычно создаются с использованием больших данных, доступных их авторам – как правило, крупным организациям. Вам такие данные вряд ли доступны. Но параметры предобученной модели могут оказаться полезны при решении вашей задачи.

Предобученную модель можно использовать двумя способами:

- 1) использовать ее параметры для инициализации своей модели;
- 2) использовать ее как экстрактор признаков для своей модели.

Использование предобученной модели в качестве инициализатора

Как отмечалось выше, выбор стратегии инициализации параметров влияет на свойства обученной модели. Предобученные модели, взятые из интернета

или обученные вами, обычно хорошо работают при решении оригинальной задачи, на которой обучались.

Если ваша задача похожа на ту, что решает предобученная модель, то велики шансы, что параметры, оптимальные для вашей текущей задачи, будут не слишком сильно отличаться от предобученных параметров, особенно в начальных слоях нейронной сети (ближайших к входу).

Обучение сети для вашей задачи, возможно, пойдет быстрее, потому что градиентный спуск будет искать оптимальные значения параметров в меньшей области потенциально хороших значений.

Если предобученная модель была построена с использованием обучающего набора, гораздо большего, чем ваш, то поиск в области потенциально хороших значений может также улучшить обобщаемость модели. Действительно, даже если какой-то аспект поведения создаваемой вами модели не отражен в обучающих примерах, то он все же мог быть «унаследован» от предобученной модели.

Использование предобученной модели в качестве экстрактора признаков

Использование предобученной модели в качестве инициализатора для своей собственной дает больше гибкости. Градиентный спуск будет модифицировать параметры во всех слоях и теоретически может достичь лучшего качества на вашей задаче. Недостаток же в том, что часто дело заканчивается обучением очень глубокой сети.

Некоторые предобученные модели содержат сотни слоев и миллионы параметров. Обучение такой большой сети может оказаться проблематичным. Безусловно, потребуется очень много вычислительных ресурсов. Кроме того, проблема исчезающего градиента в глубокой нейронной сети проявляется с большей силой, чем в сети с парой скрытых слоев.

Если ваши вычислительные ресурсы ограничены, то можно вместо этого использовать некоторые слои предобученной модели в качестве **экстракторов признаков** для своей модели. На практике это означает, что сохраняются только несколько начальных слоев предобученной модели – входной и ближайшие к нему. Их параметры «замораживаются», т. е. не подлежат изменению. Затем после замороженных слоев добавляются новые, в т. ч. и выходной слой для вашей задачи. Во время обучения на ваших данных градиентный спуск будет обновлять только параметры новых слоев.

Этот процесс показан на рис. 6.4. Голубым цветом изображена предобученная модель. Часть голубых слоев повторно используется в новой модели, а их параметры замораживаются. Зеленые слои добавлены аналитиком и обучены под решаемую задачу.

Аналитик может, если захочет, заморозить параметры всей голубой части новой сети, а обучать только параметры зеленой части. Но можно сделать несколько самых правых голубых слоев обучаемыми.

Сколько слоев предобученной модели использовать в новой? Сколько слоев замораживать? Все это решать аналитику, именно он определяет, какая архитектура будет оптимальна для конкретной задачи.

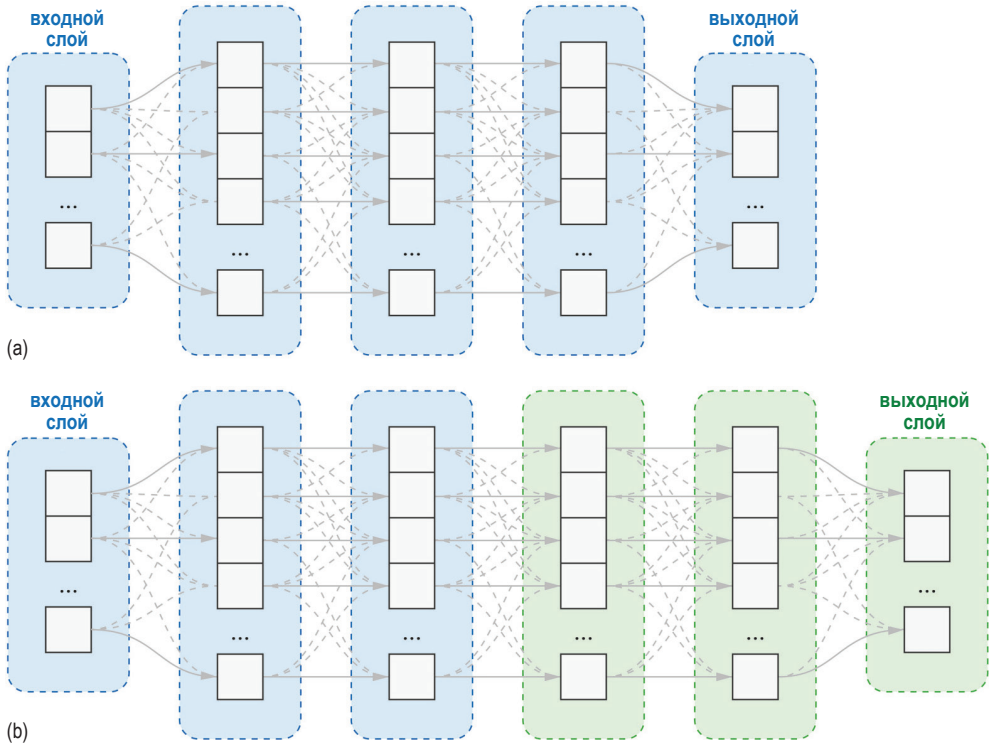


Рис. 6.4 ❖ Иллюстрация переноса обучения: (а) предобученная модель и (б) ваша модель, в которой использована левая часть от предобученной модели и добавлены новые слои, в т. ч. другой выходной слой, лучше отвечающий вашей задаче

6.2. ШТАБЕЛИРОВАНИЕ МОДЕЛЕЙ

Под **ансамблевым обучением** понимается обучение ансамблевой модели, т. е. комбинации нескольких **базовых моделей**, каждая из которых показывает худшее качество, чем их ансамбль.

6.2.1. Типы ансамблевого обучения

Существуют специальные алгоритмы ансамблевого обучения, например **обучение случайного леса** и **градиентный бустинг**. Они обучают ансамбль **слабых моделей**, насчитывающий от нескольких сотен до нескольких тысяч моделей, и получают в итоге **сильную модель**, качество которой значительно выше, чем у каждой слабой модели по отдельности. Здесь мы не будем обсуждать эти алгоритмы. Желаящие легко смогут найти их в специальной литературе по машинному обучению¹.

¹ Дополнительные сведения об алгоритмах обучения ансамблей см. в главе 7 книги «The Hundred-Page Machine Learning Book».

Почему комбинирование нескольких моделей может оказаться лучше? Потому что когда некоррелированные модели в чем-то соглашаются, велика вероятность, что это что-то – правильный исход. Ключевое слово здесь «некоррелированные». В идеале модели должны быть построены с использованием разных признаков или иметь разную природу – например, метод опорных векторов (SVM) и случайный лес. Комбинирование разных версий алгоритма обучения решающих деревьев или нескольких SVM с различными гиперпараметрами может не дать существенного улучшения качества.

Цель ансамблевого обучения – обучиться комбинировать сильные стороны базовых моделей. Существует три способа объединить слабо коррелированные модели в ансамблевую: 1) усреднение, 2) голосование и 3) штабелирование моделей.

Усреднение работает для регрессии, а также тех моделей классификации, которые возвращают оценки. Заключается оно в применении всех базовых моделей к входу x с последующим усреднением предсказаний. Чтобы понять, действительно ли усредненная модель работает лучше, чем отдельные, можно протестировать ее на контрольном наборе, применив ту или иную метрику качества.

Голосование работает для моделей классификации. Оно заключается в применении всех базовых моделей к входу x с последующим возвратом класса, за который отдано большинство голосов. Если таких классов больше одного, то можно случайно выбрать любой из них или вернуть сообщение об ошибке, если неправильная классификация могла бы нанести значительный ущерб бизнесу.

Штабелирование – метод ансамблевого обучения, при котором сильная модель обучается путем подачи ей на вход выходов более слабых моделей. Рассмотрим этот подход подробнее.

6.2.2. Алгоритм штабелирования моделей

Пусть требуется скомбинировать классификаторы f_1 , f_2 и f_3 , предсказывающие один и тот же набор классов. Чтобы создать синтетический обучающий пример (\hat{x}_i, \hat{y}_i) для штабелированной модели на основе оригинального обучающего примера (x_i, y_i) , положим $\hat{x}_i \leftarrow [f_1(x), f_2(x), f_3(x)]$ и $\hat{y}_i \leftarrow y_i$. Это показано на рис. 6.5.

Если какая-то из базовых моделей возвращает класс и оценку класса, то мы можем использовать оценки в качестве дополнительных входных признаков для штабелированной модели.

Для обучения штабелированной модели используются синтетические примеры, а ее гиперпараметры настраиваются с помощью перекрестной проверки. Проверьте, действительно ли штабелированная модель работает на контрольном наборе лучше каждой базовой модели.

Помимо использования разных алгоритмов и моделей машинного обучения, для обеспечения слабой коррелированности базовых моделей мож-

но обучать их, случайно выбирая примеры и признаки из оригинального обучающего набора. Кроме того, один и тот же алгоритм, обученный с сильно различающимися значениями гиперпараметров, теоретически может породить в достаточной степени некоррелированные модели.

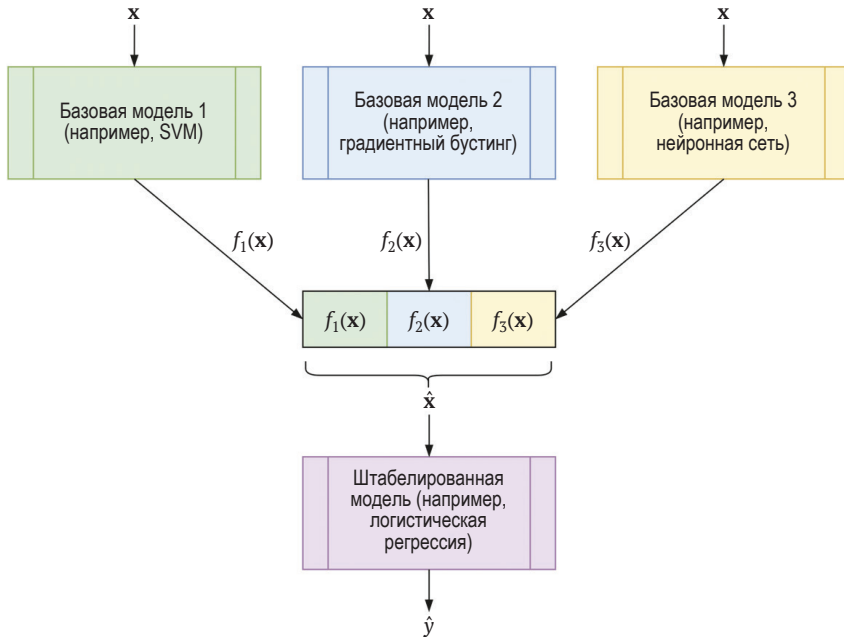


Рис. 6.5 ❖ Штабелирование трех слабо коррелированных сильных моделей

6.2.3. Просачивание данных при штабелировании моделей

Чтобы избежать **просачивания данных**, нужно соблюдать аккуратность при обучении штабелированной модели. При создании синтетического обучающего набора поступайте так же, как при перекрестной проверке. Сначала разделите все обучающие данные на десять или более групп. Чем больше групп, тем лучше, но и тем медленнее пойдет обучение модели.

Временно исключите одну группу из обучающих данных и обучите базовые модели на оставшихся группах. Затем примените базовые модели к примерам из исключенной группы. Получите предсказания и постройте синтетические обучающие примеры для исключенного блока, используя предсказания базовых моделей.

Повторите этот процесс для каждого из оставшихся блоков, в результате получится обучающий набор для штабелированной модели. Размер нового синтетического набора будет такой же, как у оригинального.

6.3. БОРЬБА СО СДВИГОМ РАСПРЕДЕЛЕНИЯ

Напомним, что зарезервированные данные должны напоминать данные, наблюдаемые в производственной среде. Но иногда их невозможно получить в достаточном количестве. В то же время у нас может быть доступ к помеченным данным, похожим на реальные, но не совсем таким. Например, мы можем располагать огромным количеством помеченных изображений, собранных веб-роботом, но наша цель – обучить классификатор на фотографиях из Instagram. Таких фотографий может быть недостаточно, поэтому мы надеемся обучить модель на данных, собранных роботом, а затем использовать ее для классификации фотографий из Instagram.

6.3.1. Обработка сдвига распределения

Когда распределения обучающих и тестовых данных не совпадают, мы говорим о **сдвиге распределения**. Обработка сдвига распределения в настоящее время – область активных исследований. Выделяют три типа сдвига распределения:

- **сдвиг ковариат** – сдвиг значений признаков;
- **сдвиг априорного распределения** – сдвиг значений целевых показателей;
- **смена концепта** – сдвиг соотношения между признаками и меткой.

Возможно, вы знаете, что для данных имеет место сдвиг распределения, но вот какого именно типа, обычно неизвестно. Если число примеров в тестовом наборе велико по сравнению с обучающим, то можно случайным образом выбрать какую-то долю тестовых примеров и перенести часть из них в обучающий набор, а часть в тестовый. Затем можно обучить модель как обычно. Однако зачастую обучающих примеров очень много, а тестовых сравнительно мало. В таком случае более эффективна **состязательная проверка** (adversarial validation).

6.3.2. Состязательная проверка

К состязательной проверке мы готовимся следующим образом. Предположим, что векторы признаков в обучающем и тестовом наборах содержат одинаковое число признаков и эти признаки представляют одну и ту же информацию. Разделим оригинальный обучающий набор на две части: обучающий набор 1 и обучающий набор 2.

Создадим модифицированный обучающий набор 1, преобразовав примеры из обучающего набора 1 следующим образом. К каждому примеру добавим оригинальную метку в качестве дополнительного признака и назначим этому примеру новую метку «Training».

Создадим модифицированный тестовый набор, преобразовав примеры из оригинального тестового набора следующим образом. К каждому примеру

добавим оригинальную метку в качестве дополнительного признака и назовем этому примеру новую метку «Test».

Объединим модифицированный обучающий набор 1 и модифицированный тестовый набор, получив в результате новый синтетический обучающий набор. Он будет использоваться для решения задачи бинарной классификации – различения примеров класса «Training» и класса «Test». С помощью синтетического обучающего набора обучим бинарный классификатор, возвращающий оценку предсказания.

Заметим, что обученный нами бинарный классификатор будет предсказывать для каждого оригинального примера, является он обучающим или тестовым. Применим этот классификатор к примерам из обучающего набора 2. Выделим те примеры, которые были предсказаны моделью как «Test» с наибольшей уверенностью. Будем использовать эти примеры в качестве контрольных данных для исходной задачи.

Удалим из обучающего набора 1 примеры, для которых бинарная модель предсказала «Training» с наибольшей уверенностью. Оставшиеся в обучающем наборе 1 примеры будем использовать в качестве обучающих данных для исходной задачи.

Оптимальный способ разделения оригинального обучающего набора на два ищется экспериментально. Также нужно определить, сколько примеров из обучающего набора 1 использовать для обучения, а сколько для контроля.

6.4. ОБРАБОТКА НЕСБАЛАНСИРОВАННЫХ НАБОРОВ ДАННЫХ

В разделе 3.9 мы рассматривали некоторые методы обработки **несбалансированных наборов данных**, в частности выборку с избытком и с недостатком и генерирование синтетических данных.

В этом разделе мы поговорим о дополнительных методах, которые применяются на этапе обучения, а не на этапах сбора и подготовки данных.

6.4.1. Взвешивание классов

Некоторые алгоритмы и модели, в частности **метод опорных векторов (SVM)**, **решающие деревья** и **случайные леса**, позволяют аналитику задавать веса каждого класса. Потеря в функции стоимости обычно умножается на вес. Аналитик может, к примеру, увеличить вес миноритарного класса. Тогда алгоритму обучения будет труднее игнорировать примеры из миноритарного класса, т. к. это привело бы к гораздо более высокой стоимости, чем без назначения весов.

Посмотрим, как это работает в случае метода опорных векторов. Задача заключается в том, чтобы различить подлинные и мошеннические транзакции в системе электронной коммерции. Примеры подлинных транзак-

ций встречаются гораздо чаще. Если использовать SVM с **мягким зазором**, то можно будет определить стоимость неправильно классифицированных примеров. Алгоритм SVM пытается передвинуть гиперплоскость, так чтобы уменьшить число неправильно классифицированных примеров. Если стоимость неправильной классификации одинакова для обоих классов, то для «мошеннических» примеров, находящихся в меньшинстве, возрастает риск неправильной классификации, поскольку это позволило бы правильно классифицировать больше примеров из мажоритарного класса. Эта ситуация показана на рис. 6.6(a). Такая проблема имеет место для большинства алгоритмов обучения, применяемых к несбалансированным данным.

Если увеличить цену неправильной классификации миноритарных примеров, то модели будет труднее классифицировать их неправильно. Но это повлечет за собой неправильную классификацию некоторых примеров мажоритарного класса, как показано на рис. 6.6(b).

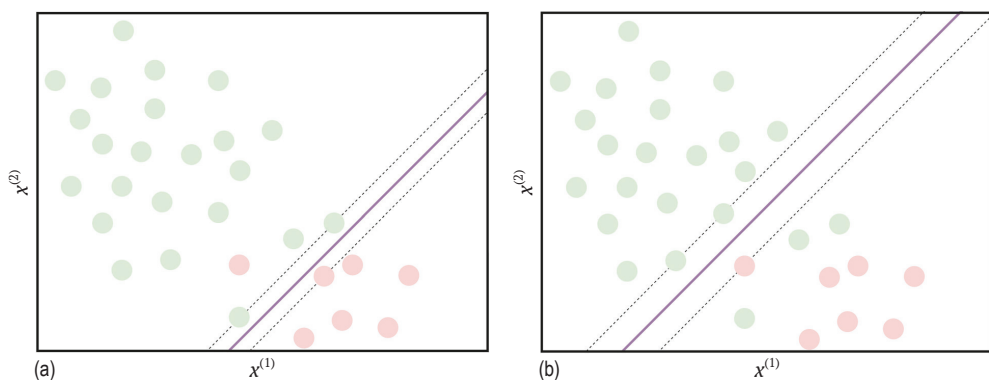


Рис. 6.6 ❖ Проблема несбалансированных данных.

(a) Веса обоих классов одинаковы.

(b) Примеры миноритарного класса имеют больший вес

6.4.2. Ансамбль перераспределенных наборов данных

Ансамблевое обучение – еще один способ сгладить остроту проблемы несбалансированных классов. Аналитик случайным образом разбивает мажоритарные примеры на H подмножеств, а затем создает H обучающих наборов. После обучения H моделей аналитик делает предсказания путем усреднения (или голосования) их выходов.

Для $H = 4$ этот процесс показан на рис. 6.7. Здесь мы преобразовали несбалансированную задачу бинарного обучения в четыре сбалансированные, разбив примеры мажоритарного класса на четыре подмножества. Примеры миноритарного класса полностью скопированы в каждое подмножество.

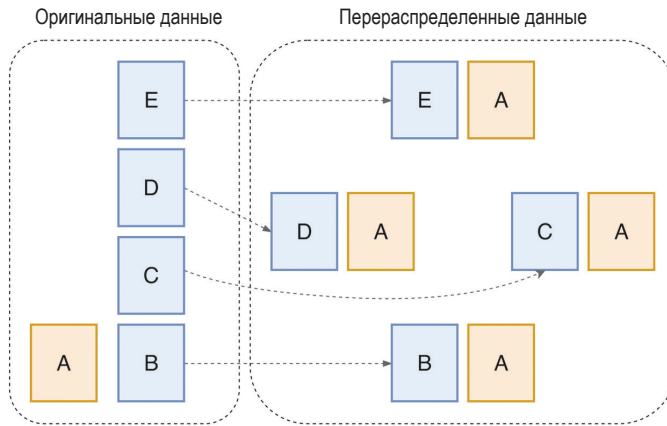


Рис. 6.7 ❖ Ансамбль перераспределенных наборов данных

Этот подход прост и хорошо масштабируется: мы можем обучать и выполнять модели на разных процессорных ядрах или узлах кластера. Кроме того, ансамбль моделей дает лучшие предсказания, чем каждая отдельная модель.

6.4.3. Другие методы

Если вы пользуетесь стохастическим градиентным спуском, то к проблеме несбалансированности классов можно подойти несколькими способами. Во-первых, можно задать разные скорости обучения для разных классов: меньше для примеров мажоритарного класса, больше – для остальных. Во-вторых, можно выполнить несколько последовательных обновлений параметров модели при каждой встрече примера миноритарного класса.

Для несбалансированных задач обучения качество модели измеряется с помощью адаптированных метрик, в частности **средней верности** и **кап-пы Коэна**, которые мы рассматривали в разделе 5.5.2 предыдущей главы.

6.5. КАЛИБРОВКА МОДЕЛИ

Иногда важно, чтобы модель классификации возвращала не только предсказанный класс, но и вероятность его правильности. Некоторые модели возвращают вместе с предсказанным классом оценку. Даже если она изменяется от 0 до 1, не всегда это вероятность.

6.5.1. Хорошо откалиброванные модели

Говорят, что модель **хорошо откалибрована**, если для входного примера x и предсказанной метки \hat{y} она возвращает оценку, которую можно интерпретировать как истинную вероятность принадлежности x к классу \hat{y} .

Например, хорошо откалиброванный бинарный классификатор должен был бы порождать оценку 0.8 приблизительно для 80 % примеров, фактически принадлежащих положительному классу.

Большинство алгоритмов машинного обучения обучают модели, не являющиеся хорошо откалиброванными, как показывают¹ **кривые калибровки** на рис. 6.8.

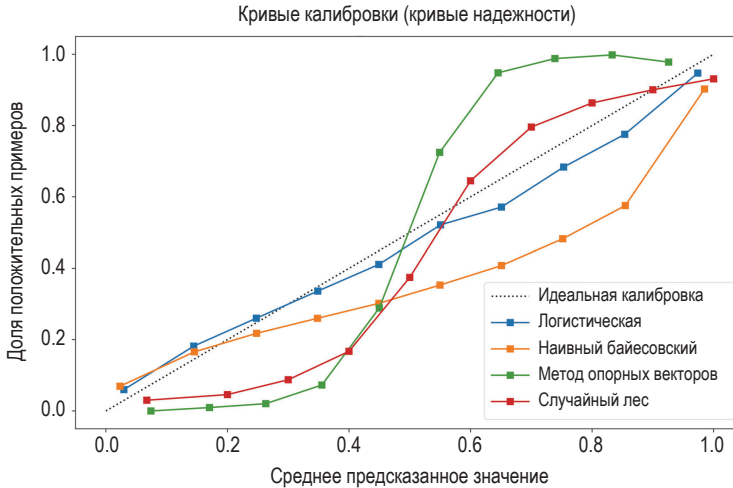


Рис. 6.8 ❖ Кривые калибровки для моделей, обученных несколькими алгоритмами машинного обучения на случайном бинарном наборе данных

Кривая калибровки для бинарной модели дает возможность понять, насколько хорошо откалибрована модель. По оси X отложены интервалы, группирующие примеры по предсказанной оценке. Например, если имеется 10 интервалов, то в самом левом находятся все примеры, для которых предсказанная оценка попадает в диапазон $[0, 0.1)$, а в самом правом — примеры с оценкой из диапазона $[0.9, 1.0]$. По оси Y отложены доли положительных примеров в каждом интервале.

Для многоклассовой классификации мы имели бы одну кривую калибровки на каждый класс типа «**один против остальных**». Это распространенная стратегия адаптации алгоритма обучения бинарного классификатора к решению многоклассовых задач. Идея в том, чтобы перейти от многоклассовой задачи к S задачам бинарной классификации и построить S бинарных классификаторов. Например, если имеется три класса $y \in \{1, 2, 3\}$, то мы создаем три копии оригинального набора данных и модифицируем их. В первой копии все метки, не равные 1, заменяются на 0. Во второй копии все метки, не равные 2, заменяются на 0. В третьей копии все метки, не равные 3, заменяются на 0. Теперь у нас имеется три задачи бинарной классификации, в которых мы хотим научиться различать метки 1 и 0, 2 и 0 и 3 и 0. Как видим,

¹ График взят со страницы <https://scikit-learn.org/stable/modules/calibration.html>.

в каждой задаче метка 0 означает «остальные» в названии метода «один против остальных».

Если модель хорошо откалибрована, то кривая калибровки колеблется вокруг диагонали (пунктирной прямой на рис. 6.8). Чем ближе кривая калибровки к диагонали, тем лучше откалибрована модель. Поскольку модель логистической регрессии возвращает истинные вероятности положительного класса, ее калибровочная кривая ближе всего к диагонали. Если модель плохо откалибрована, то кривая калибровки обычно имеет сигмоидную форму, как в моделях **опорных векторов** и **случайного леса**.

6.5.2. Методы калибровки

Для калибровки бинарной модели часто применяются два метода: **масштабирование Платта** и **изотоническая регрессия**. Оба основаны на сходных принципах.

Пусть имеется модель f , которую мы хотим откалибровать. Прежде всего нужно зарезервировать набор данных специально для калибровки. Чтобы избежать переобучения, мы не можем воспользоваться для калибровки ни обучающими, ни контрольными данными. Пусть этот калибровочный набор данных имеет размер M . Применив модель f к каждому примеру $i = 1, \dots, M$, получим для каждого i предсказание f_i . Построим новый набор данных Z , в котором каждый пример является парой (f_i, y_i) , где y_i – истинная метка примера i и метки принадлежат множеству $\{0, 1\}$.

Единственное различие между масштабированием Платта и изотонической регрессией заключается в том, что в первом случае строится модель логистической регрессии с использованием набора данных Z , а во втором – модель изотонической регрессии Z , т. е. неубывающая функция, максимально близкая к примерам. Имея откалиброванную модель z , полученную масштабированием Платта или изотонической регрессией, мы можем предсказать откалиброванную вероятность для входа x в виде $z(f(x))$.

Заметим, что откалиброванная модель может повысить качество предсказаний для конкретной задачи, но может и не повысить. Это зависит от выбранной метрики качества модели.

Согласно экспериментам¹, масштабирование Платта наиболее эффективно, когда искажение предсказанных вероятностей имеет форму сигмоиды. Изотоническая регрессия может исправить более широкий спектр искажений. К сожалению, за повышенную гибкость приходится платить. Анализ показал, что изотоническая регрессия в большей степени подвержена переобучению, а потому работает хуже масштабирования Платта, если данных мало.

Эксперименты на восьми задачах классификации также показали, что случайные леса, нейронные сети и бэггинговые решающие деревья являются наилучшими методами обучения для предсказания хорошо откалиброван-

¹ Alexandru Niculescu-Mizil and Rich Caruana. Predicting Good Probabilities With Supervised Learning // Труды 22-й международной конференции по машинному обучению, Бонн, Германия, 2005.

ных вероятностей до калибровки, но после калибровки лучшими методами являются бустинговые деревья, случайный лес и SVM.

6.6. Поиск неполадок и анализ ошибок

Искать неполадки в конвейере машинного обучения нелегко. Трудно понять, почему модель работает плохо: то ли из-за ошибки в коде, то ли из-за проблем с обучающими данными, выбором алгоритма обучения или способом проектирования конвейера. Более того, одну и ту же деградацию качества можно объяснить разными причинами. Результаты обучения могут быть чувствительны к малым изменениям гиперпараметров или состава набора данных.

Из-за этих проблем обучение модели обычно является итеративным процессом: аналитик обучает модель, смотрит на ее поведение и вносит коррективы на основе своих наблюдений.

6.6.1. Причины плохого поведения модели

Если модель плохо ведет себя на обучающих данных (недообучена), то типичные причины таковы:

- архитектура модели или алгоритм обучения недостаточно выразительны (попробуйте более сложный алгоритм, **ансамблевый метод** или более глубокую **нейронную сеть**);
- модель слишком сильно регуляризирована (уменьшите **регуляризацию**);
- выбраны неоптимальные значения гиперпараметров (настройте **гиперпараметры**);
- сконструированные признаки не обладают достаточной **предсказательной способностью** (добавьте более информативные признаки);
- данных недостаточно для обобщаемости модели (попробуйте получить больше данных, воспользуйтесь **приращением данных** или **переносом обучения**);
- в коде имеется ошибка (отладьте код определения и обучения модели).

Если модель хорошо ведет себя на обучающих данных, но плохо на зарезервированных (переобучена), то типичные причины таковы:

- данных недостаточно для обобщаемости (добавьте данные или воспользуйтесь приращением данных);
- модель недостаточно регуляризирована (усильте регуляризацию, а для нейронных сетей дополнительно воспользуйтесь **пакетной нормировкой**);
- распределения обучающих и зарезервированных данных различаются (уменьшите **сдвиг распределения**);
- выбраны неоптимальные значения гиперпараметров (настройте гиперпараметры);
- признаки обладают низкой **предсказательной способностью** (добавьте более информативные признаки).

6.6.2. Итеративное уточнение модели

Если имеется доступ к новым помеченным данным (например, вы можете пометить примеры самостоятельно или запросить помощь разметчиков), то для уточнения модели можно применить следующий простой итеративный процесс:

1. Обучить модель, используя лучшие найденные на данный момент значения гиперпараметров.
2. Протестировать модель на небольшом подмножестве контрольного набора (100–300 примеров).
3. Найти самые частые типичные ошибки на этом небольшом контрольном наборе. Исключить эти примеры из контрольного набора, потому что модель уже приспособилась к ним.
4. Сгенерировать новые признаки или добавить обучающие данные, чтобы исправить замеченные ошибки.
5. Повторять, пока не исчезнут часто встречающиеся типичные ошибки (большинство ошибок будут непохожи друг на друга).

Итеративное уточнение модели – упрощенный вариант **анализа ошибок**. Ниже описан более принципиальный подход.

6.6.3. Анализ ошибок

Ошибки бывают двух видов:

- равномерные, возникают с одинаковой частотой во всех сценариях;
- систематические, чаще возникают в сценариях определенного типа.

Систематические ошибки, для которых прослеживается закономерность, заслуживают особого внимания. Устранив причину, вы исправите ошибку сразу во многих примерах. Обычно систематические ошибки, или тенденции, возникают, когда некоторые сценарии использования плохо представлены в обучающих данных. Например, система обнаружения лиц, разработанная одним крупным производителем веб-камер, лучше работала для светлокожих, чем для чернокожих лиц. Другой пример – система обнаружения человека, оборудованная устройствами ночного видения, лучше работала днем, чем ночью, просто потому, что примеры для ночного обучения реже встречались в данных.

Совсем избежать равномерных ошибок невозможно, но важные систематические ошибки следует выявить до того, как система будет передана в эксплуатацию. Это можно сделать, кластеризовав тестовые примеры и протестировав модель на примерах из разных кластеров. Распределение производственных (онлайновых) данных может сильно отличаться от распределения офлайновых данных, которые использовались для обучения и предэксплуатационного тестирования. Поэтому кластерам, содержащим немного примеров в офлайновых данных, возможно, будут соответствовать гораздо более частые сценарии в онлайн-режиме.

В разделе 4.8 мы обсуждали несколько способов понижения размерности. Помимо кластеризации, для выявления тенденций ошибок можно исполь-

зовать метод равномерной аппроксимации и проецирования многообразий (UMAP) или **автокодировщик**. С их помощью можно понизить размерность данных до 2, а затем визуально исследовать распределение ошибок в наборе данных.

Точнее, мы можем визуализировать данные на двумерной диаграмме рассеяния, раскрасив примеры разных классов в разные цвета. Для выявления тенденций можно использовать разные маркеры в зависимости от того, было предсказание модели правильным или нет. Например, кружочками обозначать примеры с правильно предсказанной меткой, а квадратиками – остальные. Так можно будет наглядно показать области, в которых модель работает неудовлетворительно. При работе с непосредственно воспринимаемыми данными, например изображениями или текстом, полезно также визуально изучить некоторые примеры из таких областей.

Не важно, удовлетворены вы качеством модели на зарезервированных данных или нет, всегда можно улучшить модель, проанализировав отдельные ошибки. Как уже отмечалось, лучшего всего работать итеративно, рассматривая по 100–300 примеров за раз. Ограничив число обрабатываемых примеров, мы сможем быстро завершать итерации, переобучая модель после каждой, но при этом рассмотреть достаточно примеров для выявления очевидных закономерностей.

Как решить, стоит ли тратить время на исправление выявленной закономерности? Можно положить в основу решения **частоты паттернов ошибок**. Посмотрим, как это работает. Пусть верность модели равна 80 %, т. е. частота ошибок составляет 20 %. Если исправить все замеченные систематические ошибки, то качество модели можно будет улучшить самое большее на 20 %. Если в малом пакете для анализа ошибок было 300 примеров, то модель сделала $0.2 \times 300 = 60$ ошибок. Проанализируйте их одну за другой и попытайтесь понять, какие особенности входных данных стали причиной неправильной классификации этих 60 примеров. Для конкретики предположим, что наша задача – обнаружить пешеходов на изображениях улиц. Допустим, что в 60 из 300 случаев модель не смогла обнаружить пешехода. Тщательный анализ выявил два паттерна: 1) в 40 примерах изображение было размыто, 2) в 5 примерах фотография была сделана в ночное время. Следует ли тратить время на решение обеих проблем?

Решив проблему размытых изображений (например, добавив дополнительные помеченные размытые изображения в состав обучающих данных), мы можем рассчитывать, что ошибка уменьшится на $(40/60) \times 20 = 13$ %. В лучшем случае после решения этой проблемы ошибка составит $20 - 13 = 7$ %, это заметное уменьшение по сравнению с первоначальными 20 %.

С другой стороны, решив проблему ночных изображений, мы можем рассчитывать только на уменьшение ошибки на $5/60 \times 20 = 1.7$ %. Так что в лучшем случае модель станет допускать ошибки на $20 - 1.7 = 18.3$ % примеров, что для одних задач существенно, а для других не очень. Плата за сбор дополнительных помеченных ночных изображений может оказаться настолько велика, что овчинка не стоит выделки.

Чтобы исправить систематическую ошибку, можно применить один из следующих приемов или их комбинацию:

- подвергнуть вход предобработке (например, удалить фон из изображений, исправить грамматические ошибки в тексте);
- прирастить данные (например, размыть или кадрировать изображения);
- пометить дополнительные обучающие примеры;
- сконструировать новые признаки, которые позволили бы алгоритму обучения различать «трудные» случаи.

6.6.4. Анализ ошибок в комплексных системах

Допустим, что мы работаем над комплексной системой классификации документов, которая состоит из трех сцепленных моделей, как на рисунке ниже.

Пусть верность системы в целом составляет 73 %. Для бинарной классификации верность 73 % не кажется высокой. С другой стороны, если модель классификации (правый блок на рис. 6.9) поддерживает тысячи классов, то такая верность не кажется слишком низкой. Но в некоторых реальных ситуациях пользователь может ожидать от системы качества, сравнимого или даже превосходящего человека.

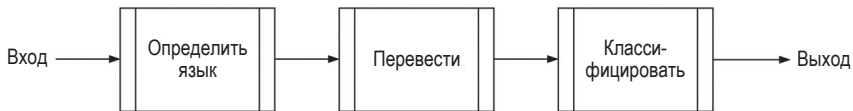


Рис. 6.9 ❖ Комплексная система классификации документов

Допустим, что эксперты со стороны заказчика ожидают, что верность системы классификации документов будет выше 73 %. Чтобы получить максимальную отдачу от дополнительных усилий, мы должны решить, какую часть системы улучшать в первую очередь.

Когда решения о чем-то принимаются на нескольких связанных уровнях, как на рис. 6.9, и не зависят друг от друга, значения верности перемножаются. Например, если верность предиктора языка была равна 95 %, верность модели машинного перевода¹ 90 %, а верность классификатора 85 %, то при условии, что все три модели независимы, общая верность трехступенчатой системы будет равна $0.95 \times 0.90 \times 0.85 = 0.73$, или 73 %. На первый взгляд, очевидно, что максимальный выигрыш получится при максимизации верности третьей модели – классификатора. Однако на практике некоторые ошибки данной модели могут вносить несущественный вклад в общее качество системы. Например, если предиктор языка часто путает испанский и португальский, то модель машинного перевода все равно сможет сгенерировать адекватный перевод для модели классификации на третьей ступени.

Работая над классификатором третьей ступени, мы можем прийти к выводу, что достигли максимального качества, так что продолжать не имеет

¹ Измерить ошибку системы машинного перевода трудно, потому что перевод редко бывает полностью верным или полностью неверным. Вместо этого применяются такие оценки, как BLEU (Bilingual Evaluation Understudy Score).

смысла. Но тогда какую из двух предыдущих моделей, предиктор языка или переводчик, следует улучшить, чтобы повысить качество системы в целом?

Определить верхнюю границу потенциала всей системы позволяет, в частности, **анализ ошибок по частям**. Заменяем предсказания одной модели заведомо верными метками, например проставленными человеком. Затем оцениваем качество всей системы. Например, вместо того чтобы использовать на второй ступени систему машинного перевода, мы можем попросить профессионального переводчика перевести текст с предсказанного языка (если язык был предсказан правильно) или сохранить оригинальный текст (если язык определен неправильно).

Допустим, мы запросили у профессионала перевод ста текстов. Теперь можно измерить, как идеальный перевод влияет на качество системы в целом. Допустим, что верность всей системы стала равна 74 %. Стало быть, потенциальный выигрыш от улучшения перевода в общем качестве системы составляет всего 1 %. Оказывается, что грандиозная задача – достичь качества человека в системе машинного перевода – не стоит потраченных усилий, поскольку в итоге мы улучшаем качество системы в целом всего на один процентный пункт. Поэтому лучше потратить время на построение более качественного предиктора языка на первой ступени, если это позволит сильнее повысить общее качество системы.

6.6.5. Использование расслоенных метрик

Если модель предполагается применять к разным сегментам, то тестировать ее следует отдельно для каждого сегмента. Например, модель, предсказывающая платежеспособность заемщиков, должна быть одинаково верной для мужчин и женщин. Чтобы добиться этого, мы можем разбить контрольные данные на несколько подмножеств, по одному на каждый сегмент. А затем вычислить метрику качества, применяя модель к каждому подмножеству.

Альтернативно можно отдельно оценить модель на каждом классе, применяя метрики точности и полноты. Напомним, что эти метрики определены только для бинарной классификации. Выделив один класс в задаче многоклассовой классификации и пометив остальные классы меткой «Другое», мы сможем по отдельности вычислить точность и полноту для каждого класса.

Заметив, что метрики качества разнятся для сегментов или классов, мы можем попытаться решить проблему, добавив помеченные данные в те сегменты или классы, где качество модели неудовлетворительно, или сконструировав дополнительные признаки.

6.6.6. Исправление неправильных меток

Человек может назначить обучающим примерам неправильные метки. Это может стать причиной плохого качества модели как на обучающих, так и на зарезервированных данных. Действительно, если похожим примерам назначены конфликтующие метки – иногда правильные, иногда нет, – то алгоритм может обучиться предсказывать неправильную метку.

Опишем простой способ выявить примеры с неправильными метками. Применим модель к обучающим данным, по которым она была построена, и проанализируем, для каких примеров она давала предсказания, отличные от меток, поставленных человеком. Если мы видим, что предсказания правительны, то изменяем метки.

При наличии времени и ресурсов можно также изучить предсказания, для которых оценка близка к порогу принятия решений. Они тоже часто имеют неправильные метки.

Если неправильные метки в обучающих данных – серьезная проблема, то ее можно решить, попросив нескольких человек назначить метки одному и тому же примеру. Пример принимается, только если все назначили ему одинаковые метки. В ситуациях, где такая строгость не требуется, можно принимать метку, если ее назначило большинство разметчиков.

6.6.7. Нахождение дополнительных примеров для пометки

Выше мы отмечали, что в результате анализа ошибок может оказаться, что необходимы дополнительные помеченные данные для некоторых областей пространства признаков. Непомеченных примеров может быть в избытке. Но как решить, какие из них следует пометить, чтобы максимизировать повышение качества модели?

Если модель возвращает оценку предсказания, то было бы эффективно использовать лучшую из имеющихся моделей для оценки непомеченных примеров. А затем пометить те примеры, для которых оценка предсказания близка к порогу принятия решения.

Если анализ ошибок выявил закономерности с помощью средств визуализации, то выбирайте примеры, окруженные большим количеством примеров с ошибками предсказания.

6.6.8. Поиск неполадок при глубоком обучении

Чтобы избежать проблем при обучении глубокой модели, действуйте, как показано на блок-схеме (рис. 6.10).

По возможности начинайте с малого, например постройте простую модель с помощью высокоуровневой библиотеки типа **Keras**. Ее должно быть очень легко проконтролировать визуально, в идеале она должна уместиться на двух экранах, не более.

Можно также воспользоваться готовой архитектурой с открытым исходным кодом, доказавшей свою работоспособность (обращайте внимание, по какой лицензии распространяется код!). Начните с:

- небольшого нормированного набора данных, помещающегося в памяти;
- самого простого в использовании оптимизатора функции стоимости (например, **Adam**);

- стратегии инициализации (например, **случайной нормальной**);
- значений по умолчанию для чувствительных гиперпараметров оптимизатора функции стоимости и слоев;
- отсутствия регуляризации.

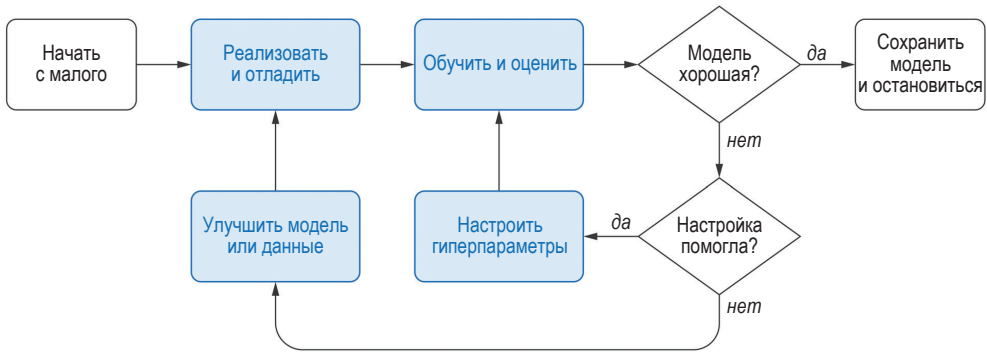


Рис. 6.10 ❖ Блок-схема поиска неполадок при глубоком обучении

Выбрав упрощенную архитектуру модели и получив набор данных, временно еще уменьшите обучающий набор данных до размера **мини-пакета**. Затем начинайте обучение. Удостоверьтесь, что упрощенная модель способна к **переобучению** на этом обучающем мини-наборе. Если переобучения не случилось, это верный признак того, что с кодом или данными что-то не так. Посмотрите, нет ли перечисленных ниже проявлений¹ и соответствующих вероятных причин.

Таблица 6.2. Типичные проблемы, из-за которых нейросетевая модель не переобучается на мини-пакете, и их типичные причины

Проявление	Вероятные причины
Ошибка растет	Неправильный знак в функции потерь или в градиенте Слишком велика скорость обучения Softmax взята не по тому измерению
Ошибка растет взрывообразно	Численная неустойчивость Слишком велика скорость обучения
Ошибка колеблется	Данные или метки повреждены (например, обнулены или неправильно перемешаны) Слишком велика скорость обучения
Ошибка выходит на плато	Слишком мала скорость обучения Градиент протекает не по всей модели Слишком сильная регуляризация Некорректные аргументы функции потерь Данные или метки повреждены

¹ Заимствовано из лекции «Troubleshooting Deep Neural Networks», прочитанной Джошем Тобином и др. в январе 2019 года.

Добившись переобучения модели на мини-пакете, вернитесь ко всему набору данных. Обучайте, оценивайте и настраивайте гиперпараметры, пока удастся получить какое-то улучшение на контрольных данных.

Если качество модели по-прежнему неудовлетворительно, измените модель (например, увеличьте глубину или ширину) или обучающие данные (например, примените другой способ предобработки или добавьте новые признаки). Отладьте изменение, снова добившись переобучения на мини-пакете, затем обучите, оцените и настройте новую модель. Продолжайте итерации, пока не станете довольны качеством.

В процессе поиска наилучшей архитектуры модели удобно не только использовать уменьшенный обучающий набор, но и упростить саму задачу. Для этого:

- создайте простой синтетический обучающий набор или
- уменьшите число классов либо разрешение входных изображений (или фрагментов видео), размер текстов, скорость передачи звука и т. д.

На шаге оценивания в блок-схеме на рис. 6.10 проверьте, не вызвано ли плохое качество модели одной из причин, перечисленных в разделе 6.6.1. Выбирайте следующий шаг в зависимости от того, как можно улучшить качество модели: путем настройки гиперпараметров либо изменения модели, признаков или обучающих данных.

6.7. РЕКОМЕНДАЦИИ

В этом разделе я собрал практические советы по обучению моделей. Это не строгие предписания, а, скорее, рекомендации, которые часто экономят время и силы и могут повысить качество результатов.

6.7.1. Поставляйте хорошую модель

Что такое хорошая модель? Хорошая модель обладает двумя свойствами:

- ее качество отвечает требованиям в соответствии с метрикой качества;
- ее безопасно использовать в производственной среде.

Модель безопасно использовать, если она удовлетворяет следующим условиям:

- она не «роняет» объемлющую систему и не вызывает ошибок при загрузке или когда ей на вход подаются плохие или неожиданные данные;
- она не потребляет слишком много ресурсов (CPU, GPU или оперативной памяти).

6.7.2. Доверяйте популярным реализациям с открытым исходным кодом

Современные библиотеки и модули для машинного обучения с открытым исходным кодом на популярных языках программирования и платформах –

Python, Java, .NET и др. – содержат эффективные реализации популярных алгоритмов, соответствующие требованиям индустрии. Обычно они распространяются по достаточно либеральной лицензии. Кроме того, существуют библиотеки и модули специально для обучения нейронных сетей.

Разрабатывать собственные алгоритмы машинного обучения считается оправданным, только если вы используете экзотический или совсем новый язык программирования. Кроме того, программировать с нуля можно, если модель предполагается выполнять в среде с очень ограниченными ресурсами или нужно такое быстродействие, которого не может обеспечить ни одна из имеющихся реализаций.

Старайтесь не использовать в одном проекте несколько языков программирования. Это увеличит стоимость тестирования, развертывания и сопровождения. Кроме того, станет труднее передавать владение проектом от одного сотрудника другому.

6.7.3. Оптимизируйте важную для бизнеса меру качества

Алгоритмы обучения стараются уменьшить ошибку на обучающих данных. А аналитик данных хочет минимизировать ошибку на тестовых данных. Однако у клиента или работодателя другая цель – оптимизировать **важную для бизнеса метрику качества**.

Минимизировав частоту ошибок на контрольном наборе, займитесь настройкой гиперпараметров, так чтобы оптимизировать важную для бизнеса метрику, даже если это приведет к увеличению частоты ошибок на контрольном наборе.

6.7.4. При обновлении начинайте с нуля

После развертывания в производственной среде некоторые модели необходимо периодически обновлять с учетом новых данных, чтобы адаптироваться к потребностям пользователя. Новые обучающие данные следует автоматически собирать с помощью скриптов (эта тема обсуждалась в разделе 3.12, посвященном **воспроизводимости**).

Всякий раз, как данные обновляются, гиперпараметры следует настраивать с нуля. В противном случае использование новых данных со старыми гиперпараметрами может привести к снижению качества модели.

Некоторые модели, например нейронные сети, можно обновлять итеративно. Но избегайте **теплого старта**, т. е. итеративного обновления существующей модели путем использования только новых обучающих примеров и прогона дополнительных итераций обучения.

Кроме того, частое обновление модели без повторного обучения с нуля может привести к **катастрофическому забыванию**. Так называется ситуация, в которой модель, которая когда-то обладала определенной способностью, теперь «забыла» ее, поскольку обучилась чему-то новому.

Заметим, что обновление модели – не то же самое, что **перенос обучения**. Аналитики применяют перенос обучения, когда недоступны данные или вычислительные ресурсы, использованные при построении предобученной модели.

6.7.5. Избегайте каскадов коррекций

Возможно, у вас есть модель m_A , решающая задачу A , но нужно получить решение m_B немного отличающейся задачи B . Возникает искушение использовать выход m_A как вход m_B и обучить m_B только на малой выборке примеров, «корректирующих» выход m_A в интересах решения задачи B . Такая техника **каскадирования коррекций** не рекомендуется.

Каскадирование коррекций не дает возможности обновить модель m_A , не обновив также модель m_B (и последующие части каскада). Какое влияние изменение m_A окажет на m_B , предсказать невозможно, но, скорее всего, ничего хорошего не будет. Кроме того, разработчик модели m_B может ничего не знать об изменении m_A , а разработчик m_A может не знать о том, что от нее зависит m_B . Негативное влияние изменений m_A на m_B может в течение длительного времени оставаться незаметным.

Вместо того чтобы строить каскад коррекций, рекомендуется обновить модель m_A , включив в нее сценарии, возникающие при решении задачи B . Было бы разумно добавить признаки, позволяющие модели отличать примеры для задачи B . Можно также воспользоваться переносом обучения или построить совершенно независимую модель для решения задачи B .

6.7.6. Используйте каскадирование моделей с осторожностью

Важно отметить, что **каскадирование моделей** не всегда плохо. Использование выхода некоторой модели как одного из многих входов для другой модели – обычное дело. Это может заметно сократить время выхода на рынок. Однако каскадирование следует использовать с осторожностью, потому что обновление одной модели в каскаде должно сопровождаться обновлением всех моделей в этом каскаде, что в конечном итоге может дорого обойтись.

Чтобы сгладить негативный эффект каскадирования моделей, можно порекомендовать две стратегии.

1. Проанализировать поток информации в программной системе и обновить или заново обучить всю цепочку. Обновленный выход модели m_A должен быть отражен в обучающих данных для модели m_B .
2. Контролировать, кто может и кто не может обращаться к модели m_A , чтобы предотвратить создание этой проблемы необъявленными потребителями. Как отмечают инженеры Google¹: «В отсутствие препятствий инженеры естественно будут использовать самый удобный сигнал из

¹ Hidden Technical Debt in Machine Learning Systems / Sculley et al. (2015).

имеющихся под рукой, особенно работая в условиях напряженного графика».

Кроме того, выход модели не должен быть простым числом или строкой. Он должен сопровождаться информацией о породившей его модели и об условиях потребления.

6.7.7. Пишите эффективный код, компилируйте и распараллеливайте

Быстрый и эффективный код позволяет ускорить обучение на порядок по сравнению с неэффективным скриптом, который вы написали на коленке во время экспериментов с единственной целью – «чтобы заработало». Современные наборы данных велики, поэтому завершения предобработки иногда приходится ждать часами, а то и днями. А обучение может занимать дни или даже недели.

При написании кода всегда помните об эффективности, даже если кажется, что некоторая функция, метод или скрипт будут вызываться нечасто. Бывает так, что код, который предполагалось запускать однократно, на самом деле вызывается в цикле миллионы раз.

Избегайте циклов. Например, если требуется вычислить **скалярное произведение** двух векторов или умножить матрицу на вектор, пользуйтесь быстрыми и эффективными функциями, имеющимися в научных библиотеках и модулях. Примерами таких библиотек на Python могут служить **NumPy** и **SciPy**. Их создавали опытные и талантливые программисты и ученые. Они опираются на низкоуровневые языки программирования типа C, а также на средства аппаратного ускорения, поэтому работают молниеносно.

По возможности компилируйте код перед выполнением. Такие библиотеки, как **PyPy** и **Numba** для Python или **pqR** для R, компилируют исходный код в машинный, непосредственно исполняемый ОС (операционной системой), что может значительно ускорить обработку данных и обучение модели.

Еще один важный аспект – распараллеливание. В современных библиотеках и модулях можно встретить алгоритмы обучения, задействующие многоядерные процессоры. Некоторые позволяют использовать GPU для ускорения обучения нейросетей и многих других моделей. Обучение некоторых моделей, например SVM, невозможно эффективно распараллелить. Но и тогда можно использовать многоядерный CPU, чтобы выполнять несколько экспериментов параллельно. Запускайте по одному эксперименту для каждой комбинации гиперпараметров, каждого географического региона или сегмента пользователей. Проводите перекрестную проверку на всех группах параллельно.

Если возможно, используйте для хранения данных твердотельные диски (SSD). Применяйте распределенные вычисления; некоторые реализации алгоритмов обучения предназначены для выполнения в распределенной среде, например Spark. Старайтесь помещать все необходимые данные в оперативную память ноутбука или сервера. В наши дни аналитики нередко работают

на серверах с 512 гигабайтами и даже с одним или двумя терабайтами оперативной памяти.

Сократив до минимума время обучения модели, вы сможете потратить больше времени на ее настройку, проверку различных идей, касающихся предобработки данных, конструирование признаков, поиск подходящей архитектуры нейронной сети и другую творческую деятельность. Наибольшую пользу проекту машинного обучения может принести прикосновение и интуиция человека. Чем больше вы, человек, будете работать, а не тупо ждать результатов, тем больше шансов на успешное завершение проекта.

Сведите **связующий код** к минимуму. Так поступают инженеры Google. Исследователи в области машинного обучения стараются разрабатывать универсальные решения в виде автономных пакетов. Их множество – и с открытым исходным кодом, и внутрифирменных, и проприетарных, и на облачных платформах. Применение универсальных пакетов часто приводит к паттерну проектирования «система со связующим кодом», в которой приходится писать много вспомогательного кода для получения данных от универсального пакета и передачи ему данных.

В долгосрочной перспективе связующий код обходится дорого. Система оказывается зависимой от особенностей конкретного пакета. Тестирование альтернатив может стать запредельно дорогим. Такое применение общего пакета тормозит усовершенствования. Становится труднее воспользоваться зависящими от предметной области свойствами или настроить целевую функцию и достичь проблемно-зависимой цели. Может оказаться, что зрелая система содержит (максимум) 5 % кода машинного обучения и (минимум) 95 % связующего кода. Бывает, что дешевле создать чистое решение с нуля, чем использовать готовый универсальный пакет.

Для борьбы со связующим кодом полезно обернуть пакеты машинного обучения, представленные черными ящиками, общими API, используемыми в вашей организации. Такая инфраструктура допускает повторное использование и снижает стоимость изменения пакетов.

Рекомендуется научиться свободно писать по крайней мере на двух языках программирования: один для быстрого создания прототипов (например, Python), другой для быстрого выполнения (например, C++). Современные языки типа Go, Kotlin и Julia могут подойти для обеих целей, но на момент написания этой книги для них еще не существовало такой же развитой экосистемы проектов машинного обучения, как для конкурентов.

6.7.8. Тестируйте на старых и новых данных

Если вы использовали для создания обучающего, контрольного и тестового наборов данные, выгруженные некоторое время назад, то наблюдайте, как модель ведет себя с данными, собранными до и после этого периода. Если поведение радикально ухудшилось, значит, имеется проблема.

Среди наиболее вероятных причин – **просачивание данных** и **сдвиг распределения**. Напомним, что просачивание данных имеет место, когда информация, недоступная в будущем или в прошлом, использовалась для

конструирования признака. Сдвиг распределения возникает, когда свойства данных изменяются со временем.

6.7.9. Больше данных лучше, чем более умный алгоритм

Столкнувшись с недостаточным качеством модели, аналитики часто испытывают соблазн придумать более изощренный алгоритм или конвейер обучения.

Однако на практике лучших результатов зачастую можно добиться, получив больше данных, а точнее больше помеченных примеров. При хорошо спроектированном процессе пометки разметчик может производить тысячи обучающих примеров каждый день. Это может оказаться дешевле, чем платить за опыт и знания, необходимые для изобретения более продвинутого алгоритма машинного обучения.

6.7.10. Новые данные лучше более изощренных признаков

Если, несмотря на добавление обучающих примеров и конструирование хитроумных признаков, качество модели все равно застряло на плато, подумайте о других источниках информации.

Например, если вы хотите предсказать, понравится ли пользователю U некоторая новость, попытайтесь добавить историческую информацию о нем в качестве признаков. Или кластеризуйте всех пользователей и используйте информацию о k ближайших соседях U в качестве новых признаков. Это проще, чем программировать очень сложные признаки или хитроумно комбинировать имеющиеся признаки.

6.7.11. Радуйтесь крохотным достижениям

Много мелких улучшений модели могут дать желаемый результат быстрее, чем поиск одной революционной идеи.

Кроме того, пробуя разные идеи, аналитик лучше узнает данные, что в конечном итоге может натолкнуть его на искомую революционную идею.

6.7.12. Обеспечьте воспроизводимость

Большинство алгоритмов машинного обучения стохастические. Например, обучая нейронную сеть, мы инициализируем параметры модели случайным образом; при стохастическом градиентном спуске мини-пакеты генерируются случайно; случайно строятся решающие деревья в случайном лесу; при разделении данных на три набора примеры перемешиваются случайным

образом и т. д. Это означает, что при обучении модели на одних и тех же данных два раза мы можем получить две разные модели. Чтобы обеспечить воспроизводимость, рекомендуется задавать **случайное начальное значение**, инициализирующее генератор псевдослучайных чисел. Если не изменять начальное значение, то при обучении на одних и тех же данных мы будем получать одну и ту же модель.

Случайное начальное значение можно задать методом `np.random.seed(15)` (в NumPy и scikit-learn), `tf.random.set_seed(15)` (в TensorFlow), `torch.manual_seed(15)` (в PyTorch) или `set.seed(15)` (в R). Точное значение не играет роли, лишь бы оно было постоянным.

Даже если каркас машинного обучения позволяет задать случайное начальное значение, нет никакой гарантии, что часть каркаса, в которой используется рандомизация, не изменится при переходе к следующей версии. Для обеспечения воспроизводимости все зависимости каждого проекта должны быть изолированы. Сделать это можно разными способами: либо с помощью таких инструментов, как **virtualenv** в Python и **Packrat** в R, либо ставя эксперименты в стандартных **виртуальных машинах** или **контейнерах**. О виртуализации мы будем подробнее говорить в разделе 8.3.

При поставке модели не забывайте включить все имеющуюся информацию о **воспроизводимости**. Помимо описания набора данных и признаков, в частности документации и метаданных (см. разделы 3.11 и 4.11), каждая модель должна сопровождаться документацией, содержащей следующие сведения:

- спецификация всех гиперпараметров, включая диапазоны изменения и значения по умолчанию;
- какой метод применялся для выбора наилучших гиперпараметров;
- определение конкретной меры качества или статистики, использованных для оценивания моделей-кандидатов, и значение этого показателя для наилучшей модели;
- описание вычислительной инфраструктуры;
- среднее время работы каждой обученной модели и оценка стоимости обучения.

6.8. РЕЗЮМЕ

У стратегии обучения глубоких моделей больше аспектов, чем для поверхностной модели. В то же время она лучше обоснована теоретически и более пригодна для автоматизации.

Вместо того чтобы обучать модель с нуля, бывает полезно начать с предобученной модели. Организации, имеющие доступ к большим данным, обучили и выложили в открытый доступ очень глубокие нейронные сети с архитектурами, оптимизированными для задач обработки изображений или текстов на естественном языке.

Предобученную модель можно использовать двумя способами: 1) использовать ее обученные параметры для инициализации параметров своей мо-

дели; 2) использовать ее в качестве экстрактора признаков для своей модели. Применение предобученной модели для построения своей собственной называется переносом обучения. Тот факт, что глубокие модели допускают перенос обучения, – одно из самых важных свойств глубокого обучения.

Стохастический градиентный спуск на мини-пакетах со всеми своими вариантами – наиболее употребительный алгоритм оптимизации **функции стоимости** для глубоких моделей.

Алгоритм обратного распространения вычисляет частные производные по всем параметрам глубокой модели, применяя правило дифференцирования сложной функции. В каждой эпохе градиентный спуск обновляет все параметры, используя частные производные. Скорость обучения контролирует важность обновления. Процесс продолжается до достижения сходимости – состояния, в котором значения параметров перестают существенно изменяться после эпохи обучения. Тогда алгоритм останавливается.

Существует несколько популярных усовершенствований алгоритма стохастического градиентного спуска на мини-пакетах, например Momentum, RMSProp и Adam. Эти алгоритмы изменяют скорость обучения автоматически, основываясь на характеристиках процесса обучения. Нам не нужно выбирать начальное значение скорости обучения, план и темп ее снижения и значения прочих гиперпараметров. Эти алгоритмы продемонстрировали хорошее качество на практике, поэтому специалисты часто используют их, вместо того чтобы настраивать скорость обучения вручную.

Помимо L1- и L2-регуляризации, к нейронным сетям можно с пользой применить другие виды регуляризаторов: прореживание, раннюю остановку и пакетную нормировку. Прореживание – простой, но очень эффективный метод регуляризации. Применять пакетную нормировку настоятельно рекомендуется.

Под ансамблевым обучением понимается обучение ансамбля моделей, т. е. комбинации нескольких базовых моделей, каждая из которых работает хуже, чем ансамбль в целом. Существуют алгоритмы ансамблевого обучения, например случайный лес и градиентный бустинг, которые строят ансамбль, насчитывающий от нескольких сотен до нескольких тысяч слабых моделей, и получают на выходе сильную модель, демонстрирующую гораздо более высокое качество, чем у слабых моделей.

Сильные модели можно объединить в ансамбль, усреднив их выходы (в случае регрессии) или проведя голосование (в случае классификации). Штабелирование моделей – самый эффективный из ансамблевых методов – подразумевает обучение метамодели, которая принимает выходы базовых моделей в качестве входов.

Помимо выборки с недостатком и с избытком, проблемы несбалансированного обучения можно решить, применив назначение весов классам и ансамбль перераспределенных наборов данных. Если ваша модель обучается методом стохастического градиентного спуска, то несбалансированность классов можно компенсировать еще двумя способами: 1) задавать разные скорости обучения для разных классов; 2) производить несколько последовательных обновлений параметров модели всякий раз, как встречается пример миноритарного класса.

Для несбалансированных задач обучения качество модели измеряется с помощью адаптированных метрик качества, например средней верности и каппы Козна.

Искать неполадки в конвейере машинного обучения нелегко. Трудно понять, почему модель работает плохо: то ли из-за ошибки в коде, то ли из-за проблем с обучающими данными, выбором алгоритма обучения или способом проектирования конвейера. Кроме того, обучение может быть чувствительно к малым изменениям гиперпараметров или состава набора данных.

Модель машинного обучения может допускать ошибки двух видов: равномерные, которые возникают с одинаковой частотой во всех сценариях, и систематические, чаще возникающие в сценариях определенного типа.

Систематические ошибки заслуживают особого внимания, потому что, устранив причину, мы сможем исправить ошибки сразу во многих примерах.

Качество модели можно итеративно улучшать с помощью следующего простого процесса:

1. Обучить модель, используя лучшие найденные на данный момент значения гиперпараметров.
2. Протестировать модель на небольшом подмножестве контрольного набора (100–300 примеров).
3. Найти самые частые типичные ошибки на этом небольшом контрольном наборе. Исключить эти примеры из контрольного набора, потому что модель уже приспособилась к ним.
4. Сгенерировать новые признаки или добавить обучающие данные, чтобы исправить замеченные ошибки.
5. Повторять, пока не исчезнут часто встречающиеся типичные ошибки (большинство ошибок будут непохожи друг на друга).

В комплексных системах машинного обучения анализ ошибок производится по частям. Сначала мы заменяем предсказания одной модели точными метками (например, проставленными человеком) и смотрим, как повысилось качество системы в целом. Если существенно, то следует уделить больше внимания улучшению этой конкретной модели.

Для обеспечения воспроизводимости задавайте случайное начальное значение и не забывайте прикладывать к модели всю необходимую информацию.

Глава 7

Оценивание модели

Статистические модели играют все более важную роль в современной организации. Будучи применена в деловом контексте, модель может повлиять на финансовые показатели организации. Но может также провоцировать риск наступления ответственности. Поэтому любую модель, работающую в производственной среде, следует тщательно и непрерывно оценивать.

Оценивание модели – пятый этап жизненного цикла проекта машинного обучения.

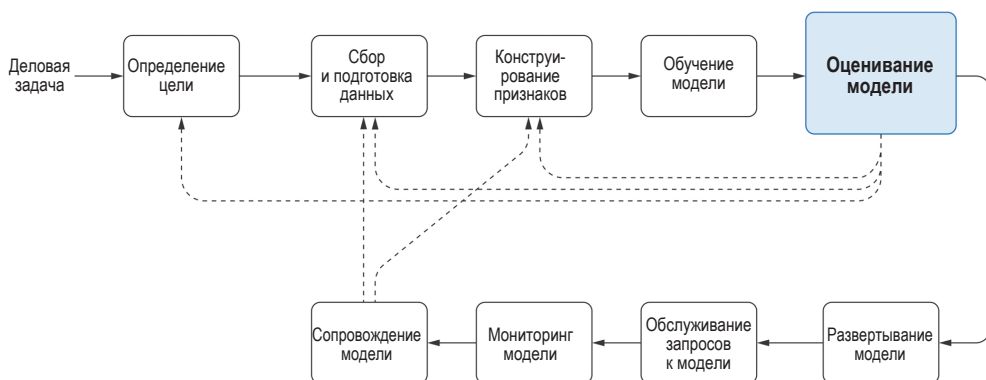


Рис. 7.1 ❖ Жизненный цикл проекта машинного обучения

В зависимости от области применения модели, а также бизнес-целей и ограничений организации оценивание модели может включать следующие задачи.

- Оценка правовых рисков, связанных с эксплуатацией модели. Например, некоторые предсказания модели могут косвенно сообщать конфиденциальную информацию. Киберпреступники и конкуренты могут попытаться подвергнуть обучающие данные модели обратной разработке. Кроме того, при использовании для предсказания некоторые признаки, например возраст, пол или раса, могут стать причиной для обвинения организации в пристрастности или даже дискриминации.
- Изучение основных свойств распределений обучающих и производственных данных. Сравнивая статистические распределения примеров, признаков и меток, можно выявить сдвиг распределения. Существ-

венное различие между распределениями говорит о необходимости обновить обучающие данные и заново обучить модель.

- Оценивание качества модели. Прежде чем развертывать модель в производственной среде, необходимо оценить ее предсказательную способность на внешних данных, которые не использовались для обучения. Внешние данные должны включать как исторические, так и онлайнные примеры, взятые из производственной среды. Контекст оценивания на онлайнных данных, получаемых в режиме реального времени, должен быть близок к производственной среде.
- Мониторинг качества развернутой модели. Со временем качество модели может снижаться. Важно, чтобы это было вовремя замечено, после чего нужно либо обновить модель, добавив новые данные, либо обучить совершенно другую модель. Процесс мониторинга модели должен быть тщательно спланирован и автоматизирован, хотя может подразумевать и участие человека. Более подробно мы рассмотрим его в главе 9.

В этой главе мы рассмотрим *некоторые* трюки, на которые пускаются статистики при оценивании модели. Инженерия машинного обучения – развивающаяся дисциплина, на некоторые вопросы еще нет устоявшихся и простых в применении ответов. В частности, оценивание представлено с точки зрения инженера, хотя у каждой компании есть собственные уникальные критерии успеха. Прежде чем оценивать решение, очень важно удостовериться, что специально обученные люди выполнили наиболее трудную часть проекта: установили, как выглядит успех и какие вопросы нужно задать, сформулировав все это в виде специфических для компании метрик и целевых функций.

Частая причина провала проекта заключается в том, что инженеры отвечают на удобные вопросы, пользуясь стандартными инструментами, а не на правильные вопросы, для которых разработаны специальные инструменты – что может потребовать консультаций с профессиональным статистиком, после того как руководители проекта и все заинтересованные стороны внесли свою лепту. Отметим, что некоторые методы, описанные в этой главе, а особенно используемые в составе А/В-тестирования (раздел 7.2), приведены только в качестве примеров и могут быть неприменимы к вашей конкретной деловой задаче. В важных крупномасштабных проектах было бы ошибкой делать все самостоятельно. Нужно сотрудничать с руководством и советоваться со статистиками.

7.1. ОФЛАЙНОВОЕ И ОНЛАЙНОВОЕ ОЦЕНИВАНИЯ

В разделе 5.5 мы рассматривали методы, применяемые при **офлайновом оценивании модели**. Оно применяется в процессе обучения модели аналитиком. Аналитик пробует разные признаки, модели, алгоритмы и гиперпараметры. Чтобы обучение модели шло в правильном направлении, используются такие инструменты, как матрица неточностей и различные метрики

качества: точность, полнота и площадь под кривой (AUC), позволяющие сравнивать разные модели.

Сначала производится оценка выбранной метрики качества и сравнение моделей на контрольных данных. Затем качество лучшей модели еще раз оценивается, также в офлайн-режиме, на тестовых данных. Эта последняя оценка гарантирует качество модели после передачи в эксплуатацию. В этой главе мы, среди прочего, поговорим о том, как задавать статистические границы офлайн-качества модели на тестовых данных.

Значительная часть этой главы посвящена **онлайн-оцениванию модели**, т. е. тестированию и сравнению моделей в производственном режиме с использованием онлайн-данных. Различие между офлайн- и онлайн-оцениваниями модели, а также место обоих типов оценивания в системе машинного обучения показаны на рис. 7.2.

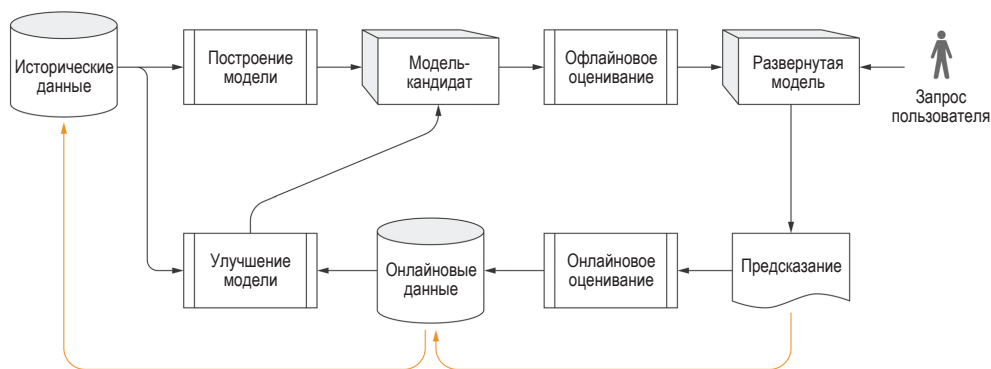


Рис. 7.2 ❖ Место офлайн- и онлайн-оцениваний модели в системе машинного обучения

На рис. 7.2 исторические данные сначала используются для обучения модели-кандидата на развертывание. Затем эта модель оценивается в офлайн-режиме, и если результат удовлетворителен, то кандидат развертывается и начинает принимать запросы пользователей. После этого запросы и предсказания модели используются для ее онлайн-оценивания. На онлайн-данных производится улучшение модели. И чтобы замкнуть петлю, онлайн-данные постоянно копируются в репозиторий офлайн-данных.

Зачем оценивать модель и в офлайн- и в онлайн-режимах? Оценивание в офлайн-режиме показывает, удалось ли аналитику найти правильные признаки, алгоритм обучения, модель и значения гиперпараметров. Иными словами, офлайн-модель говорит, насколько хороша модель с инженерной точки зрения.

С другой стороны, онлайн-оценивание сосредоточено на измерении результатов, важных для бизнеса: удовлетворенности пользователей, средним временем пребывания на сайте, открываемости полученных сообщений и кликабельности. Эта информация не всегда отражается в исторических

данных, но именно она интересует компанию. Кроме того, бывает так, что в офлайн-режиме невозможно протестировать работу модели в определенных условиях, которые наблюдаются в онлайн-режиме, например: разрыв соединения, потеря данных, задержки вызова.

Результаты оценки качества, полученные на исторических данных, будут актуальны и после развертывания, только если распределение данных не меняется со временем. Но на практике так бывает не всегда. К типичным примерам **сдвига распределения** можно отнести постоянно изменяющиеся интересы пользователя мобильного или онлайн-приложения, нестабильность финансовых рынков, изменение климата или изменение механических свойств системы, свойства которой должна предсказывать модель.

Отсюда следует, что модель необходимо постоянно мониторить после развертывания в производственной среде. Когда распределение сдвигается, модель следует модифицировать с учетом новых данных и развернуть заново. Один из способов мониторинга – сравнение качества модели на онлайн-данных и исторических данных. Если качество на онлайн-данных заметно ухудшилось, значит, настало время обучить модель заново.

Существуют разные формы онлайн-оценивания, служащие разным целям. Например, мониторинг во время выполнения проверяет, отвечает ли работающая модель предъявляемым требованиям.

Еще один типичный сценарий – мониторинг поведения пользователей в ответ на разные версии модели. Одна из популярных техник такого рода – А/В-тестирование. Мы разбиваем пользователей системы на две группы, А и В. Эти группы обслуживаются старой и новой моделями соответственно. После этого применяется статистический критерий значимости, чтобы определить, лучше качество новой модели или хуже.

Многорукие бандиты – еще одна популярная техника онлайн-оценивания модели. Как и А/В-тестирование, она определяет лучшие модели, предлагая разные модели-кандидаты какой-то части пользователей. Постепенно, благодаря сбору информации о качестве, доля пользователей, которым предлагается лучшая модель, увеличивается – и так до тех пор, пока оценка не станет надежной.

7.2. А/В-ТЕСТИРОВАНИЕ

А/В-тестирование – один из самых часто используемых статистических методов. Применительно к онлайн-оцениванию модели он позволяет отвечать на вопросы вида «Работает ли новая модель m_B в производственном режиме лучше, чем имеющаяся модель m_A ?» или «Какая из двух моделей-кандидатов лучше работает в производственном режиме?».

А/В-тестирование часто применяется на веб-сайтах и в мобильных приложениях, чтобы проверить, как конкретное изменение в дизайне или выборе слов влияет на бизнес-метрики типа заинтересованности пользователей, кликабельности или темпа продаж.

Допустим, что нам интересно, стоит ли заменять имеющуюся (старую) модель, которая сейчас эксплуатируется, новой моделью. Реальный трафик, содержащий входные данные для модели, разбивается на две непересекающиеся группы: А (контрольная) и В (экспериментальная). Трафик группы А направляется старой модели, а группы В – новой модели.

По результатам сравнения качества обеих моделей принимается решение о том, какая модель лучше. Для сравнения качества применяется **проверка статистических гипотез**.

В общем случае в теории проверки статистических гипотез рассматриваются **нулевая гипотеза** и **альтернативная гипотеза**. Цель А/В-тестирования обычно формулируется следующим образом: «Приводит ли новая модель к статистически значимому изменению конкретной бизнес-метрики?» Нулевая гипотеза утверждает, что новая модель не изменяет среднего значения бизнес-метрики.

А/В-тест – не просто критерий, а семейство критериев. В зависимости от бизнес-метрики качества применяются различные статистические инструменты. Но принцип разбиения пользователей на две группы и измерения статистической значимости различий между метриками для обеих групп остается неизменным.

Описание всех формулировок А/В-тестов выходит за рамки этой книги. Мы рассмотрим только две из них, но зато применимые к широкому кругу практических ситуаций.

7.2.1. G-критерий

Первая формулировка А/В-теста основана на **G-критерии**. Она подходит для метрики, которая подсчитывает ответы на вопрос, предполагающий ответ «да» или «нет». Преимущество G-критерия в том, что вопрос может быть любым, лишь бы на него можно было дать только два ответа. Вот несколько примеров вопросов:

- купил ли пользователь рекомендованный товар?
- потратил ли пользователь более 50 долларов за месяц?
- возобновил ли пользователь подписку?

Посмотрим, как это применяется на практике. Мы хотим решить, работает ли новая модель лучше старой. Для этого сформулируем вопрос, требующий ответа да-нет, который определит метрику. Затем случайным образом разобьем пользователей на две группы, А и В. Трафик пользователей из группы А направляется в среду, где работает старая модель, а пользователей из группы – на новую модель. Наблюдаем за действиями каждого пользователя и регистрируем ответы как «да» или «нет». Заполняем таблицу, представленную на рис. 7.3.

В этой таблице \hat{a}_{yes} – число пользователей из группы А, давших на вопрос ответ «да», \hat{b}_{yes} – число пользователей из группы В, давших ответ «да», \hat{a}_{no} – число пользователей из группы А, давших ответ «нет» и т. д., $n_{yes} = \hat{a}_{yes} + \hat{b}_{yes}$, $n_{no} = \hat{a}_{no} + \hat{b}_{no}$, $n_a = \hat{a}_{yes} + \hat{a}_{no}$, $n_b = \hat{b}_{yes} + \hat{b}_{no}$, $n_{total} = n_{yes} + n_{no} = n_a + n_b$.

	Да	Нет	
А	\hat{a}_{yes}	\hat{a}_{no}	n_a
В	\hat{b}_{yes}	\hat{b}_{no}	n_b
	\hat{n}_{yes}	\hat{n}_{no}	n_{total}

Рис. 7.3 ❖ Ответы да-нет,
полученные от пользователей из группы А и В

Вычислим ожидаемое число ответов «да» и «нет» для групп А и В, т. е. сколько тех и других ответов мы получили бы, если бы модели А и В были эквивалентны.

$$\begin{aligned}
 a_{yes} &\stackrel{\text{def}}{=} n_a \frac{n_{yes}}{n_{total}}, \\
 a_{no} &\stackrel{\text{def}}{=} n_a \frac{n_{no}}{n_{total}}, \\
 b_{yes} &\stackrel{\text{def}}{=} n_b \frac{n_{yes}}{n_{total}}, \\
 b_{no} &\stackrel{\text{def}}{=} n_b \frac{n_{no}}{n_{total}}.
 \end{aligned} \tag{7.1}$$

Теперь найдем значение G -критерия¹:

$$G \stackrel{\text{def}}{=} 2 \left(\hat{a}_{yes} \ln \left(\frac{\hat{a}_{yes}}{a_{yes}} \right) + \hat{a}_{no} \ln \left(\frac{\hat{a}_{no}}{a_{no}} \right) + \hat{b}_{yes} \ln \left(\frac{\hat{b}_{yes}}{b_{yes}} \right) + \hat{b}_{no} \ln \left(\frac{\hat{b}_{no}}{b_{no}} \right) \right).$$

G – это мера различия выборок из А и В. С точки зрения статистики, если верна нулевая гипотеза (А и В эквивалентны), то G имеет **распределение хи-квадрат** с одной степенью свободы:

$$G \sim \chi_1^2.$$

Иначе говоря, если бы А и В были эквивалентны, то G должно было бы быть мало. Большое значение G заставляет подозревать, что одна модель лучше другой. Например, допустим, что $G = 3.84$. Если бы А и В были эквивалентны (т. е. верна нулевая гипотеза), то вероятность наблюдать $G \geq 3.84$ приближенно равна 5 %. Часто эту вероятность называют p -значением.

Если p -значение достаточно мало (скажем, меньше 0.05), то качество новой модели и качество старой, скорее всего, различны (нулевая гипотеза отвергается). При этом если b_{yes} больше a_{yes} , то новая модель, вероятно, лучше старой, а в противном случае лучше старая модель.

¹ О том, как выводится эта формула, можно прочитать в учебнике математической статистики или в Википедии.

Если p -значение, соответствующее значению G , недостаточно мало, то наблюдаемое различие в качестве между новой и старой моделями статистически не значимо, так что можно продолжать эксплуатацию старой модели.

Вычислять p -значение G -критерия удобно на любом языке программирования. На Python это можно сделать следующим образом:

```
1 from scipy.stats import chi2
2 def get_p_value(G):
3     p_value = 1 - chi2.cdf(G, 1)
4     return p_value
А на R – так:
1 get_p_value <- function(G) {
2     p_value <- pchisq(G, df=1, lower.tail=FALSE)
3     return(p_value)
4 }
```

Статистический результат G -критерия имеет смысл, если в каждой группе имеется как минимум 10 ответов «да» и «нет», хотя эту оценку следует принимать с долей скептицизма. Если тестирование обходится не слишком дорого, то должно хватить примерно 1000 ответов в каждой из двух групп, при условии что имеется по меньшей мере 100 ответов каждого типа. Отметим, что общее число ответов в группах может быть различно.

Если получить не менее 100 ответов каждого типа в каждой группе стоит дорого, то можно воспользоваться аппроксимацией p -значения очень похожего критерия, воспользовавшись **моделированием методом Монте-Карло**.

На R код выглядит так:

```
1 p_value <- chisq.test(x,
2     simulate.p.value = TRUE)$p.value
3 }
```

где x – таблица сопряженности размера 2×2 , показанная на рис. 7.3.

Заметим, что можно тестировать и более двух моделей (например, А, В и С) и более двух ответов на вопрос, определяющий метрику (например, «да», «нет», «может быть»). Если мы хотим тестировать k моделей и l возможных ответов, то G -статистика будет иметь распределение хи-квадрат с $(k-1) \times (l-1)$ степенями свободы. Проблема в том, что тест с несколькими моделями и ответами скажет, отличаются ли модели, но не скажет, в чем отличие. На практике проще сравнивать текущую модель с одной новой и формулировать определяющий метрику вопрос, так чтобы на него было только два ответа. Более сложные эксперименты выходят за рамки этой книги.

Заметим, что при наличии более двух моделей возникает соблазн выполнить бинарные сравнения пар моделей с помощью теста, предназначенного для сравнения двух моделей. Это не рекомендуется, потому что с научной точки зрения может быть некорректно. Лучше проконсультироваться со статистиком.

7.2.2. Z-критерий

Вторая формулировка А/В-теста применяется, когда каждому пользователю задается вопрос «Сколько?» (а не вопрос, требующий ответа «да» или «нет»).

Вот несколько примеров таких вопросов.

1. Сколько времени пользователь провел на сайте в течение одного сеанса?
2. Сколько денег пользователь потратил за месяц?
3. Сколько новостей пользователь прочел за неделю?

Для определенности будем измерять время, потраченное пользователем на сайте, где развернута модель. Как обычно, пользователи направляются на одну из двух версий сайта: А или В, причем А обслуживается старой моделью, а В – новой. Нулевая гипотеза состоит в том, что пользователи обеих версий проводят на сайте в среднем одинаковое время, а альтернативная гипотеза – в том, что на сайте В они проводят больше времени, чем на А. Обозначим n_A число пользователей, отправленных на версию А, а n_B – число пользователей, отправленных на версию В. Пусть индексы i и j относятся к пользователям из группы А и В соответственно.

Для вычисления Z-критерия мы сначала вычисляем выборочное среднее и выборочную дисперсию для А и В. Выборочное среднее равно

$$\begin{aligned}\hat{\mu}_A &\stackrel{\text{def}}{=} \frac{1}{n_A} \sum_{i=1}^{n_A} a_i, \\ \hat{\mu}_B &\stackrel{\text{def}}{=} \frac{1}{n_B} \sum_{j=1}^{n_B} b_j,\end{aligned}\tag{7.2}$$

где a_i и b_j – время, проведенное на сайте соответственно пользователями i и j .

Выборочная дисперсия для А и В равна соответственно

$$\begin{aligned}\hat{\sigma}_A^2 &\stackrel{\text{def}}{=} \frac{1}{n_A} \sum_{i=1}^{n_A} (\hat{\mu}_A - a_i)^2, \\ \hat{\sigma}_B^2 &\stackrel{\text{def}}{=} \frac{1}{n_B} \sum_{j=1}^{n_B} (\hat{\mu}_B - b_j)^2.\end{aligned}\tag{7.3}$$

Тогда значение Z-критерия равно

$$Z \stackrel{\text{def}}{=} \frac{\hat{\mu}_B - \hat{\mu}_A}{\sqrt{\frac{\hat{\sigma}_B^2}{n_B} + \frac{\hat{\sigma}_A^2}{n_A}}}.$$

Чем больше Z, тем вероятнее, что различие между А и В значимо. Если справедлива нулевая гипотеза (т. е. А и В эквивалентны), то Z имеет приближенно стандартизованное нормальное распределение

$$Z \approx \mathcal{N}(0, 1).$$

Это так, только если размер выборки велик и $\sigma_A^2 \approx \sigma_B^2$. В противном случае лучше спросить совета у профессионального статистика.

Как и для G-критерия, мы будем использовать p -значение, чтобы решить, достаточно ли Z велико, чтобы можно было считать, что пользователи действительно проводят на В больше времени, чем на А. Чтобы вычислить p -значение, мы находим вероятность выборки из этого распределения Z -значения, которые было бы таким же экстремальным (расходящимся с нулевой гипотезой), как вычисленное Z -значение. Например, допустим, что на произведенной выборке $Z = 2.64$. Если бы А и В были эквивалентны, то вероятность наблюдать $Z \geq 2.64$ равнялась бы 5 %.

Чтобы интерпретировать результат теста, мы сравниваем p -значение с выбранным уровнем значимости. Если наш уровень значимости равен 5 %, то, получив p -значение, меньшее 0.05, мы отвергаем нулевую гипотезу, которая говорит, что различие в качестве двух моделей не является статистически значимым. Таким образом, новая модель работает лучше старой.

Если p -значение больше или равно 0.05, то мы не отвергаем нулевую гипотезу. Возможно, модели все-таки различаются, но у нас нет достаточных свидетельств в пользу этого утверждения. В таком случае оставляем старую модель, пока факты не вынудят нас поменять решение. Отсутствие фактов означает, что нужно и дальше делать, как раньше. Заметим также, что мы не можем просто продолжать сбор свидетельств, пока p -значение не окажется меньше 0.05; с научной точки зрения, это было бы некорректно. Проконсультируйтесь со статистиком и спроектируйте другой тест.

Что касается уровней значимости, то нет единого мнения о том, какой порог оптимален. На практике обычно используются значения 0.05 или 0.01. Их особенно любил законодатель мод в статистике Рональд Фишер, работавший в 1920-х годах. Можете выбрать значение больше или меньше, если считаете, что это правильно для вашего приложения. Чем меньше значение, тем больше свидетельств должно быть собрано, чтобы появились основания изменить решение.

Как и в случае G-критерия, искать p -значение Z-критерия удобно на любом языке программирования. На Python это можно сделать так:

```
1 from scipy.stats import norm
2 def get_p_value(Z):
3     p_value = norm.sf(Z)
4     return p_value
```

А на R так:

```
1 get_p_value <- function(Z) {
2     p_value <- 1-pnorm(Z)
3     return(p_value)
4 }
```

Для получения наилучших результатов рекомендуется задавать n_A и n_B равными 1000 или больше.

7.2.3. Заключительные замечания и предупреждения

В начале этой главы мы говорили, что некоторые рассматриваемые здесь методы служат лишь в качестве примеров и могут не подойти для решения конкретной задачи, стоящей перед вашей компанией. В частности, оба описанных выше статистических критерия изучаются в вузах и действительно широко используются на практике, но, к сожалению, не всегда пригодны для решения бизнес-задач. Отметив это, Кэсси Козырьков, главный специалист по теории принятия решений в Google, рецензировавшая эту главу, подчеркнула, что применение обоих критериев редко бывает оправдано на практике, потому что они только показывают, что модели различны, но не говорят, верно ли, что они различаются «по крайней мере на x ». Если замена старой модели новой стоит дорого или чревата рисками, то знать, что новая модель «в чем-то» лучше, недостаточно для принятия решения о замене. В таком случае нужно спроектировать специальный критерий для стоящей перед нами задачи, а для этого лучше всего обратиться к статистике¹.

Тщательно тестируйте программный код своего А/В-теста. Правильную оценку модели вы сможете получить, только если реализуете все как надо. В противном случае будет не понятно, что не так: тест не скажет, что он неисправен.

Кроме того, следите, чтобы измерения в группах А и В производились в одно и то же время. Помните, что трафик на сайте существенно зависит от времени суток и от дня недели. Для чистоты эксперимента не сравнивайте результаты измерений, сделанных в разное время. Эти соображения относятся и к другим возможным измеряемым параметрам, которые могут оказывать существенное влияние на поведение пользователя, как то: страна проживания, скорость интернет-соединения или версия браузера.

7.3. Многорукий бандит

Более продвинутый и зачастую предпочтительный способ онлайн-оценки и выбора модели – **многорукий бандит** (multi-armed bandit – MAB). У А/В-тестирования есть один важный недостаток. Количество результатов тестирования в группах А и В, необходимое для вычисления значения А/В-теста, велико. Значительная доля пользователей, направляемых на неоптимальную модель, будет испытывать неудобства в течение длительного времени.

В идеале мы хотели бы предъявлять пользователю неоптимальную модель как можно реже. Вместе с тем нам нужно предъявить пользователям

¹ К сожалению, в такой короткой книге, как эта, описать все особые случаи и критерии невозможно. Советую время от времени заглядывать на сопроводительный сайт книги. Возможно, там будут размещены дополнительные статистические критерии.

обе модели столько раз, сколько необходимо для получения надежных оценок качества. Это называется **дилеммой исследования и использования**: с одной стороны, мы хотим исследовать качество моделей в достаточной степени, чтобы надежно выбрать лучшую, а с другой – хотим по возможности использовать качество лучшей модели.

В теории вероятностей задачей о многоруком бандите называется задача, в которой фиксированное и ограниченное множество ресурсов должно быть распределено между конкурирующими потребителями, так чтобы максимизировать ожидаемое вознаграждение. Свойства каждого потребителя известны в момент выделения ресурсов лишь частично, но могут быть поняты лучше со временем, если мы выделим ему ресурсы.

Посмотрим, как задачу о многоруком бандите можно применить к онлайн-оцениванию двух моделей. (Для большего количества моделей подход не меняется.)

В данном случае ограниченное множество ресурсов – это пользователи системы. Конкурирующие потребители, называемые также «рычагами», – наши модели. Чтобы выделить ресурсы потребителю (иными словами, «дернуть за рычаг»), мы должны направить пользователя на версию системы, в которой работает определенная модель. Мы хотим максимизировать ожидаемое вознаграждение, определяемое бизнес-ориентированной метрикой качества. Это может быть, например, среднее время, проведенное на сайте во время сеанса, среднее число прочитанных за неделю новостей, процентная доля пользователей, купивших рекомендованную статью, и т. д.

UCB1 (Upper Confidence Bound – верхняя доверительная граница) – популярный алгоритм решения задачи о многоруком бандите. Алгоритм динамически выбирает рычаг, исходя из качества этого рычага в прошлом и из того, как много алгоритм знает о нем. Иными словами, UCB1 направляет пользователя на лучшую модель чаще, если его доверие к качеству этой модели велико. В противном случае UCB1 может направить пользователя к неоптимальной модели, чтобы получить более уверенную оценку ее качества. Когда алгоритм будет достаточно уверен в качестве каждой модели, он почти всегда будет направлять пользователей на лучшую модель.

Математически UCB1 работает следующим образом. Обозначим c_a количество раз, когда был выбран рычаг a (с момента начала эксперимента), а v_a – среднее вознаграждение, полученное при выборе этого рычага. Вознаграждение соответствует значению бизнес-ориентированной метрики качества. Для иллюстрации будем считать, что метрикой является среднее время, проведенное пользователем в системе в течение одного сеанса. Таким образом, вознаграждение за выбор рычага равно продолжительности сеанса.

В начальный момент c_a и v_a равны нулю для всех рычагов, $a = 1, \dots, M$. После выбора рычага a наблюдается вознаграждение r и c_a увеличивается на 1; затем v_a обновляется следующим образом:

$$v_a \leftarrow \frac{c_a - 1}{c_a} \times v_a + \frac{r}{c_a}.$$

На каждом временном шаге (т. е. при каждом входе нового пользователя в систему) рычаг (т. е. версия системы, на которую будет направлен пользова-

тель) выбирается следующим образом. Если $c_a = 0$ для некоторого рычага a , то он и выбирается; в противном случае выбирается рычаг с наибольшим значением UCB. Значение UCB рычага a , обозначаемое u_a , по определению, равно:

$$u_a \stackrel{\text{def}}{=} v_a + \sqrt{\frac{2 \times \log(c)}{c_a}}, \quad \text{где } c \stackrel{\text{def}}{=} \sum_a^M c_a.$$

Доказано, что этот алгоритм сходится к оптимальному решению. То есть UCB1 будет выбирать наилучший рычаг большую часть времени.

На Python код, реализующий UCB1, выглядит следующим образом:

```

1 class UCB1():
2     def __init__(self, n_arms):
3         self.c = [0]*n_arms
4         self.v = [0.0]*n_arms
5         self.M = n_arms
6         return
7
8     def select_arm(self):
9         for a in range(self.M):
10             if self.c[a] == 0:
11                 return a
12         u = [0.0]*self.M
13         c = sum(self.c)
14         for a in range(self.M):
15             bonus = math.sqrt((2 * math.log(c)) / float(self.c[a]))
16             u[a] = self.v[a] + bonus
17         return u.index(max(u))
18
19     def update(self, a, r):
20         self.c[a] += 1
21         v_a = ((self.c[a] - 1) / float(self.c[a])) * self.v[a] \
22             + (r / float(self.c[a]))
23         self.v[a] = v_a
24         return

1 setClass("UCB1", representation(count="numeric", value="numeric", M="numeric"))
2
3 setGeneric("select_arm", function(x) standardGeneric("select_arm"))
4 setMethod("select_arm", "UCB1", function(x) {
5     for (a in seq(from = 1, to = x@M, by = 1)) {
6         if(x@count[a] == 0) {
7             return(a)
8         }
9     }
10     u <- rep(0.0, x@M)
11     count <- sum(x@count)
12     for (a in seq(from = 1, to = x@M, by = 1)){
13         print(a)
14         bonus <- sqrt((2 * log(count)) / x@count[a])
15         u[a] <- x@value[a] + bonus

```

```

16   }
17   match(c(max(u)),u)
18 })
19
20 setGeneric("update", function(x, a, r) standardGeneric("update"))
21 setMethod("update", "UCB1", function(x, a, r) {
22   x@count[a] <- x@count[a] + 1
23   v_a <- ((x@count[a] - 1) / x@count[a]) * x@value[a] + (r / x@count[a])
24   x@value[a] <- v_a
25 })
26
27 UCB1 <- function(M) {
28   new("UCB1", count = rep(0, M), value = rep(0.0, M), M = M)
29 }

```

7.4. СТАТИСТИЧЕСКИЕ ГРАНИЦЫ КАЧЕСТВА МОДЕЛИ

При описании качества модели иногда требуется, помимо значения метрики, указывать статистические границы, называемые также **статистическим интервалом**.

У читателей, знакомых с другими книгами по машинному обучению или популярными блогами, может возникнуть вопрос, почему «статистический интервал», а не «доверительный интервал». Причина в том, что иногда авторы называют «доверительным интервалом» (confidence interval) то, что на самом деле является «байесовским доверительным интервалом» (credible interval). Разница между тем и другим важна, потому что эти термины имеют разный смысл в частотной и байесовской статистиках. В этой книге я решил не обременять читателя тонкими различиями между ними. Для специалиста полезно представлять себе статистический интервал следующим образом: 95%-ный статистический интервал означает, что с вероятностью 95 % оцениваемый параметр находится внутри интервала. Строго говоря, это определение байесовского доверительного интервала. Интерпретация доверительного интервала несколько иная, и большинство начинающих изучать статистику приходят к пониманию различия, только прочитав несколько учебников. Для наших целей предложенной выше интерпретации статистического интервала достаточно.

Существует несколько методов определения статистических границ модели. Одни применяются к моделям классификации, другие – к моделям регрессии. В этом разделе мы рассмотрим некоторые из этих методов.

7.4.1. Статистический интервал для ошибки классификации

Если мы включаем в описание модели классификации частоту появления ошибок «err» (где $\text{err} \stackrel{\text{def}}{=} 1 - \text{верность}$), то вычислить статистический интервал для err можно следующим образом.

Пусть N – размер тестового набора. Тогда с вероятностью 99 % err принадлежит интервалу

$$[err - \delta, err + \delta],$$

где $\delta \stackrel{\text{def}}{=} z_N \sqrt{\frac{err(1 - err)}{N}}$ и $z_N = 2.58$.

Значение z_N зависит от требуемого **уровня достоверности**. Для уровня достоверности 99 % $z_N = 2.58$. Для других уровней значения z_N приведены в таблице ниже:

Уровень достоверности	80 %	90 %	95 %	98 %	99 %
z_N	1.28	1.64	1.96	2.33	2.58

Как и в случае p -значений, удобно написать для нахождения значений z_N простенькую программу. На Python это можно сделать так:

```
1 from scipy.stats import norm
2 def get_z_N(confidence_level): # a value in (0,100)
3     z_N = norm.ppf(1-0.5*(1 - confidence_level/100.0))
4     return z_N
```

А на R – так:

```
1 get_z_N <- function(confidence_level) {# a value in (0,100)
2   z_N <- qnorm(1-0.5*(1 - confidence_level/100.0))
3   return(z_N)
4 }
```

Теоретически этот метод должен работать даже для совсем небольших тестовых наборов, когда $N \geq 30$. Но более точное эвристическое правило для нахождения минимального размера тестового набора N формулируется так: N должно быть таким, что $N \times err(1 - err) \geq 5$. Интуитивно, чем больше размер тестового набора, тем меньше неопределенность истинного качества модели.

7.4.2. Бутстреп статистического интервала

Популярный метод нахождения статистического интервала для любой метрики, применимый равно к классификации и регрессии, основан на идее **бутстреп**а. Так называется статистическая процедура, заключающаяся в генерировании B выборок из набора данных и последующем обучении или вычислении некоторой статистики на этих B выборках. В частности, эта идея лежит в основе алгоритма обучения **случайного леса**.

Вот как идея бутстрепа применяется к нахождению статистического интервала для метрики. Пусть дан тестовый набор, произведем B случайных выборок S_b , $b = 1, \dots, B$. Чтобы получить выборку S_b для некоторого b , воспользуемся **выбором с возвращением**. Это означает, что мы начинаем с пустого множества, затем случайно выбираем пример из тестового набора

и помещаем его точную копию в S_b , оставляя оригинал в тестовом наборе. Такой случайный выбор примеров с помещением в S_b продолжается, пока не окажется, что $|S_b| = N$.

Произведя B выборок из тестового набора, вычислим значение метрики качества m_b , используя каждую выборку S_b в качестве тестового набора. Отсортируем эти B значений в порядке возрастания. Затем вычислим сумму S всех B значений метрики: $S \stackrel{\text{def}}{=} \sum_{b=1}^B m_b$. Для получения s -процентного статистического интервала для этой метрики выберем наименьший интервал $[a, b]$ такой, что сумма значений m_b , попавших в него, составляет не менее s процентов S . Это и будет искомым статистический интервал.

Предыдущий абзац может показаться невнятным, поэтому проиллюстрируем на примере. Пусть $B = 10$. Допустим, что в результате применения модели к B бутстрепным выборкам получились такие значения метрики: [9.8, 7.5, 7.9, 10.1, 9.7, 8.4, 7.1, 9.9, 7.7, 8.5]. Отсортируем их в порядке возрастания: [7.1, 7.5, 7.7, 7.9, 8.4, 8.5, 9.7, 9.8, 9.9, 10.1]. Пусть уровень достоверности s равен 80 %. Тогда левой границей a статистического интервала будет 7.46, а правой границей b – 9.92. Эти два значения были найдены с помощью функции `percentile` на Python:

```
1 from numpy import percentile
2 def get_interval(values, confidence_level):
3     # confidence_level - значение в интервале (0,100)
4     lower = percentile(values, (100.0-confidence_level)/2.0)
5     upper = percentile(values, confidence_level+((100.0-confidence_level)/2.0))
6     return (lower, upper)
```

На R то же самое можно сделать с помощью функции `quantile`:

```
1 get_interval <- function(values, confidence_level) {
2     # confidence_level - значение в интервале (0,100)
3     cl <- confidence_level/100.0
4     quant <- quantile(values, probs = c((1.0-cl)/2.0, cl+((1.0-cl)/2.0)),
5                       names = FALSE)
6     return(quant)
7 }
```

Зная границы $a = 7.46$ и $b = 9.92$ статистического интервала, мы можем утверждать, что значение метрики для нашей модели лежит в интервале [7.46, 9.92], с вероятностью 80 %.

На практике аналитики выбирают в качестве уровня достоверности значение 95 % или 99 %. Чем выше уровень, тем шире интервал. Количество бутстрепных выборок B обычно полагают равным 100.

7.4.3. Бутстреп интервала предсказания для регрессии

До сих пор мы рассматривали статистический интервал для всей модели и заданной метрики качества. В этом разделе мы применим метод бутстреп

к вычислению **интервала предсказания** для модели регрессии и заданного вектора признаков \mathbf{x} , который эта модель получает на входе.

Мы хотим ответить на следующий вопрос. Если дана модель регрессии f и входной вектор признаков \mathbf{x} , то при каком интервале значений $[f_{\min}(\mathbf{x}), f_{\max}(\mathbf{x})]$ предсказание $f(\mathbf{x})$ оказывается внутри него с вероятностью s процентов?

Процедура бутстрепа аналогична. Единственная разница в том, что теперь мы генерируем B бутстрепных выборок из обучающего, а не тестового набора. Используя их в качестве B обучающих наборов, мы строим B моделей регрессии. Обозначим входной вектор признаков \mathbf{x} . Зафиксируем уровень достоверности s . Применим B моделей к \mathbf{x} и получим B предсказаний. Теперь, используя ту же технику, что и выше, найдем наименьший интервал $[a, b]$ такой, что сумма значений предсказаний, попавших в него, составляет не менее s процентов от суммы B предсказаний. Затем вернем предсказание $f(\mathbf{x})$, которое с вероятностью s процентов окажется в интервале $[a, b]$.

Как и раньше, уровень достоверности обычно берут равным 95 % или 99 %. Число бутстрепных выборок B выбирают равно 100 (или столько, сколько позволяет время).

7.5. ОЦЕНИВАНИЕ АДЕКВАТНОСТИ ТЕСТОВОГО НАБОРА

В традиционной программной инженерии тесты служат для нахождения дефектов в программах. Комплект тестов строится таким образом, чтобы ошибки можно было обнаружить до того, как программа будет передана в эксплуатацию. Тот же подход применим к тестированию всего кода, «окружающего» статистическую модель: код, получающий вход от пользователя, преобразует его в признаки, а код, интерпретирующий выход модели, отдает результат пользователю.

Однако саму модель следует подвергнуть дополнительному оцениванию. Тестовые примеры, используемые для оценивания модели, следует проектировать так, чтобы некорректное поведение модели обнаруживалось до ее развертывания в производственном режиме.

7.5.1. Нейронное покрытие

При оценивании нейронной сети, особенно такой, которую предполагается использовать в особо ответственных обстоятельствах, например в беспилотном автомобиле или космическом корабле, тестовый набор должен обладать хорошим покрытием. **Нейронное покрытие** тестового набора для нейросетевой модели определяется как отношение числа блоков (нейронов), активируемых примерами из тестового набора, к общему числу блоков. У хорошего тестового набора нейронное покрытие близко к 100 %.

Для построения такого тестового набора мы начинаем с множества непомеченных примеров, когда ни один блок модели не покрыт. Затем итеративно выполняются следующие действия:

- 1) случайным образом выбрать непомеченный пример i и пометить его;
- 2) отправить вектор признаков x_i на вход модели;
- 3) запомнить, какие блоки модели были активированы вектором x_i ;
- 4) если предсказание было правильно, пометить эти блоки как покрытые;
- 5) вернуться к шагу 1; продолжать итерации, пока нейронное покрытие не станет близко к 100 %.

Блок считается активированным, если его выход превышает некоторый порог. Для блоков типа ReLU этот порог обычно равен 0, а для логистической сигмоиды – 0.5.

7.5.2. Мутационное тестирование

В программной инженерии хорошее тестовое покрытие **тестируемой программы** (software under test – SUT) можно найти с помощью так называемого **мутационного тестирования**. Пусть имеется набор тестов, разработанных для тестирования SUT. Сгенерируем несколько «мутантов» SUT. Мутантом называется версия SUT, подвергнутая нескольким случайным модификациям, как то: замена в исходном коде знака $+$ на $-$, знака $<$ на $>$, удаление ветви `else` в предложении `if-else` и т. д. Затем применяем набор тестов к каждому мутанту и смотрим, имеется ли хотя бы один непрошедший тест. Говорят, что мутант убит, если для него хотя бы один тест не проходит. Далее вычисляется доля убитых мутантов во всем множестве мутантов. Для хорошего набора тестов эта доля равна 100 %.

В машинном обучении можно применить похожий подход. Но чтобы создать мутантную статистическую модель, нужно модифицировать не код, а обучающие данные. Если модель глубокая, то можно также случайным образом удалить или добавить слой либо удалить или заменить функцию активации. Для модификации обучающих данных мы можем:

- добавить примеры-дубликаты;
- фальсифицировать метки некоторых примеров;
- удалить некоторые примеры;
- прибавить случайный шум к значениям некоторых признаков.

Говорят, что мутант убит, если эта мутантная статистическая модель дает неверное предсказание хотя бы на одном тестовом примере.

7.6. ОЦЕНИВАНИЕ СВОЙСТВ МОДЕЛИ

Измеряя качество модели в соответствии с некоторой метрикой, например верностью или AUC, мы оцениваем свойство **корректности**. Но, помимо этого очевидно необходимого свойства модели, иногда требуется оценивать и другие, например робастность и справедливость.

7.6.1. Робастность

Под **робастностью** модели машинного обучения понимается устойчивость качества модели после добавления шума к входным данным. Робастная модель должна демонстрировать следующее поведение. Если к входному примеру прибавить возмущение – случайный шум, то качество модели должно уменьшиться пропорционально уровню шума.

Рассмотрим входной вектор признаков \mathbf{x} . До применения модели f к этому примеру модифицируем значения некоторых случайно выбранных признаков, заменив их нулями, и получим в результате модифицированный вход \mathbf{x}' . Продолжим случайно выбирать и заменять значения признаков в \mathbf{x} , пока **евклидово расстояние** между \mathbf{x} и \mathbf{x}' остается меньше некоторого порога δ . Затем применим модель f к \mathbf{x} и \mathbf{x}' и получим предсказания $f(\mathbf{x})$ и $f(\mathbf{x}')$. Зафиксируем значения δ и ε . Говорят, что модель ε -робастна к δ -возмущению входа, если для любых \mathbf{x} и \mathbf{x}' таких, что $\|\mathbf{x} - \mathbf{x}'\| \leq \delta$, имеет место неравенство $|f(\mathbf{x}) - f(\mathbf{x}')| \leq \varepsilon$.

Если имеется несколько моделей, одинаково хороших с точки зрения метрики качества, то для развертывания в производственной среде лучше выбрать ту, которая ε -робастна на тестовых данных с наименьшим ε . Но на деле не всегда понятно, как выбрать подходящее значение δ . Поэтому на практике лучше выявлять робастную модель среди нескольких кандидатов следующим образом.

Будем говорить, что тестовый набор является δ -возмущенным, если он получен применением δ -возмущения ко всем примерам исходного тестового набора. Выберем модель f , которую хотим проверить на робастность. Зададим разумное значение $\hat{\varepsilon}$ такое, что если предсказание модели в производственной среде отличается от правильного предсказания не более чем на $\hat{\varepsilon}$, то мы готовы признать модель приемлемой. Начнем с небольшого значения δ и построим δ -возмущенный набор данных. Найдём такое минимальное ε , что для любого примера \mathbf{x} из исходного тестового набора и соответствующего ему \mathbf{x}' из δ -возмущенного тестового набора имеет место неравенство $|f(\mathbf{x}) - f(\mathbf{x}')| \leq \varepsilon$.

Если $\varepsilon \geq \hat{\varepsilon}$, то мы выбрали слишком большое значение δ ; уменьшим его и повторим все сначала.

Если $\varepsilon < \hat{\varepsilon}$, то немного увеличим δ , построим δ -возмущенный тестовый набор, найдём для него ε и продолжим увеличивать δ , пока ε остается меньше $\hat{\varepsilon}$. Найдя значение $\delta = \hat{\delta}$, для которого $\varepsilon \geq \hat{\varepsilon}$, заметим, что тестируемая модель f является $\hat{\varepsilon}$ -робастной к $\hat{\delta}$ -возмущению входа. Теперь выберем другую модель, которую хотим проверить на робастность, и найдём для нее $\hat{\delta}$; продолжаем, пока не проверим все модели.

Зная значение δ -возмущения для каждой модели, развернем в производственной среде ту, для которой $\hat{\delta}$ наибольшее.

7.6.2. Справедливость

Алгоритмы машинного обучения обучаются тому, чему их учат люди. Обучение производится на обучающих примерах. У людей есть пристрастия, которые мо-

гут повлиять на сбор и пометку данных. Иногда следы пристрастности можно встретить в исторических, культурных или географических данных. А это, как мы видели в разделе 3.2 главы 3, может привести к смещенным моделям.

Атрибуты, которые нуждаются в защите от несправедливости, называются **защищенными**, или **чувствительными**. Примерами юридически признанных защищенных атрибутов являются раса, цвет кожи, пол, религия, национальность, гражданство, возраст, беременность, семейное положение, статус ветерана и генетическая информация.

Справедливость зачастую определяется предметной областью, и в каждой области могут быть свои правила и установления. К регулируемым предметным областям относятся кредитные отношения, образование, найм на работу, жилищное строительство и общественные места.

Определение справедливости сильно зависит от предметной области. На момент написания этой книги в научно-технической литературе не существовало единодушного мнения о том, что такое справедливость. К наиболее часто цитируемым концепциям относятся демографический паритет и равенство возможностей.

Демографический паритет (называемый также **статистическим паритетом**) означает, что все сегменты защищенного атрибута получают от модели положительные предсказания с одинаковой частотой.

Предположим, что положительное предсказание означает «возможность поступления в университет» или «получение займа». Математически демографический паритет определяется следующим образом. Пусть G_1 и G_2 – две непересекающиеся группы, принадлежащие тестовым данным, разделенные по чувствительному атрибуту j , скажем полу. Пусть $\mathbf{x}^{(j)} = 1$, если \mathbf{x} представляет женщину, и $\mathbf{x}^{(j)} = 0$ в противном случае. Тестируемая бинарная модель f удовлетворяет условию демографического паритета, если $\Pr(f(\mathbf{x}_i) = 1 | \mathbf{x}_i \in G_1) = \Pr(f(\mathbf{x}_k) = 1 | \mathbf{x}_k \in G_2)$. То есть по результатам измерения на тестовых данных вероятность, что модель f предскажет 1 для женщин, такая же, как вероятность предсказать 1 для мужчин.

Исключение защищенных атрибутов из вектора признаков в обучающих данных не гарантирует, что модель будет обладать свойством демографического паритета, поскольку некоторые из оставшихся признаков могут **коррелировать** с исключенными.

Равенство возможностей означает, что все группы получают от модели положительные предсказания с одинаковой частотой в предположении, что лица из этой группы соответствуют требованиям.

Математически тестируемая бинарная модель f удовлетворяет условию равных возможностей, если $\Pr(f(\mathbf{x}_i) = 1 | \mathbf{x}_i \in G_1 \text{ и } y_i = 1) = \Pr(f(\mathbf{x}_k) = 1 | \mathbf{x}_k \in G_2 \text{ и } y_k = 1)$, где y_i и y_k – фактические метки векторов признаков \mathbf{x}_i и \mathbf{x}_k соответственно. Это равенство означает, что по результатам измерения на тестовых данных вероятность, что модель f предскажет 1 для женщин, отвечающих требованиям для такого предсказания, такая же, как вероятность предсказать 1 для мужчин, также отвечающих необходимым требованиям. В терминах **матрицы неточностей** равенство возможностей означает, что **частота истинно положительных результатов** (TPR) должна быть одинакова для каждого значения защищенного атрибута.

7.7. РЕЗЮМЕ

Все статистические модели, работающие в производственной среде, должны тщательно и непрерывно оцениваться.

В зависимости от предметной области модели, а также целей и ограничений организации оценивание модели включает следующие задачи:

- оценка правовых рисков, связанных с эксплуатацией модели;
- изучение основных свойств распределений данных, применявшихся для обучения модели;
- оценивание качества модели до ее развертывания;
- мониторинг качества развернутой модели.

Офлайновое оценивание производится после обучения модели. Оно основано на исторических данных. Онлайновое оценивание заключается в тестировании и сравнении моделей в производственной среде на онлайновых данных.

Популярный метод онлайнового оценивания – А/В-тестирование. При выполнении А/В-тестирования мы разбиваем пользователей на две группы, А и В, которые обслуживаются соответственно старой и новой моделями. Затем мы применяем статистический критерий значимости, чтобы решить, действительно ли новая модель статистически отличается от старой.

Многорукий бандит – еще один популярный метод онлайнового оценивания моделей. Вначале мы случайным образом предъявляем все модели пользователям. Затем доля предъявляемых моделей низкого качества постепенно уменьшается, пока не останется единственная, наилучшая модель, обслуживающая большинство запросов.

Помимо метрик качества, мы должны включить в документацию по модели статистические границы, или статистический интервал.

Для моделей классификации и регрессии статистический интервал для любой метрики можно вычислить с помощью широко распространенной техники бутстрепа. Эта статистическая процедура подразумевает генерирование B выборок из набора данных, после чего производится обучение модели и вычисление некоторой статистики на данных из каждой выборки.

Тестовые примеры, использованные для оценивания модели, должны обеспечивать обнаружение некорректного поведения, прежде чем модель будет передана в эксплуатацию. Для оценивания тестового набора можно применять такие методы, как нейронное покрытие и мутационное тестирование.

Если модели предполагается использовать в особо ответственной системе или в области, регулируемой законодательством (например, кредитные отношения, образование, занятость, найм на работу, жилищное строительство и общественные места), то необходимо оценивать ее верность, робастность и справедливость.

Глава 8

Развертывание модели

После того как модель построена и тщательно протестирована, ее можно развертывать. Развертывание означает, что модель может принимать запросы от пользователей производственной системы. Приняв запрос, система преобразует его в вектор признаков, который передается модели для оценки. Результат возвращается пользователю.

Развертывание модели – шестой этап жизненного цикла проекта машинного обучения.

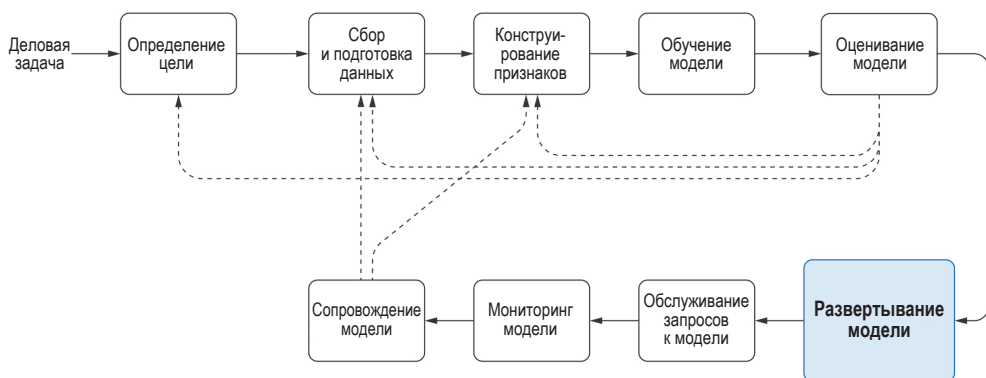


Рис. 8.1 ❖ Жизненный цикл проекта машинного обучения

Обученную модель можно развернуть разными способами: на сервере или на устройстве пользователя, сразу для всех пользователей или только для небольшой группы. Ниже мы рассмотрим все варианты.

Перечислим **паттерны развертывания** модели:

- статически, как часть устанавливаемого программного пакета;
- динамически на устройстве пользователя;
- динамически на сервере;
- потоком.

8.1. СТАТИЧЕСКОЕ РАЗВЕРТЫВАНИЕ

Статическое развертывание модели машинного обучения очень похоже на традиционное развертывание программ. Мы готовим установочный двоичный файл всей программной системы. Модель включается в пакет как ресурс, доступный во время выполнения. В зависимости от операционной системы и среды выполнения объекты модели и экстрактора признаков можно предоставлять в виде динамически компокуемой библиотеки (DLL в Windows), разделяемого объекта (so-файл в Linux) или сериализовать и сохранить в виде стандартного ресурса для систем на основе виртуальных машин, например Java и .Net.

У статического развертывания много достоинств:

- программа имеет прямой доступ к модели, поэтому запросы пользователей исполняются быстро;
- данные пользователей не нужно загружать на сервер в момент предсказания, это экономит время и обеспечивает конфиденциальность;
- модель можно вызывать, когда пользователь не подключен к сети;
- поставщику программного обеспечения не нужно поддерживать модель в работоспособном состоянии, эта ответственность возлагается на пользователя.

Однако есть и недостатки. Первый и самый главный – не всегда очевидно, где проходит граница между кодом машинного обучения и кодом приложения. Поэтому становится труднее обновить модель, не обновляя все приложение целиком. Второй – если для оценки модель предъявляет определенные требования к вычислительным возможностям (например, доступ к аппаратному ускорителю или GPU), то возрастает сложность и не всегда легко понять, допустимо ли статическое развертывание.

8.2. ДИНАМИЧЕСКОЕ РАЗВЕРТЫВАНИЕ НА УСТРОЙСТВЕ ПОЛЬЗОВАТЕЛЯ

Динамическое развертывание на устройствах похоже на статическое развертывание в том смысле, что пользователь исполняет часть системы как приложение на своем устройстве. Разница в том, что модель не является частью двоичного кода приложения. Так обеспечивается лучшее разделение обязанностей. Обновление модели производится без обновления всего приложения, работающего на устройстве пользователя. Кроме того, при динамическом развертывании один и тот же код может выбрать разные модели, исходя из доступных вычислительных ресурсов.

Динамическое развертывание можно осуществить несколькими способами:

- развернуть параметры модели;
- развернуть сериализованный объект;
- развернуть в браузере.

8.2.1. Развертывание параметров модели

В этом сценарии развертывания файл модели содержит только обученные параметры, а среда выполнения модели устанавливается на устройство пользователя. Некоторые пакеты машинного обучения, например **TensorFlow**, имеют облегченную версию, способную работать на мобильных устройствах. С другой стороны, такие каркасы, как **Core ML** от Apple, позволяют исполнять модели, созданные с помощью таких популярных пакетов, как **scikit-learn**, **Keras** и **XGBoost**, на устройствах Apple.

8.2.2. Развертывание сериализованного объекта

В этом случае файл модели представляет собой сериализованный объект, который приложению предстоит десериализовать. Плюс в том, что на устройстве пользователя не нужно иметь среду выполнения модели. Все необходимые зависимости будут десериализованы вместе с объектом модели.

Очевидный недостаток – в том, что обновление может оказаться довольно «тяжелым», а если у вашей системы миллионы пользователей, то это проблема.

8.2.3. Развертывание в браузере

У большинства современных устройств – как настольных, так и мобильных – есть доступ к браузеру. Некоторые каркасы машинного обучения, например **TensorFlow.js**, имеют версии, позволяющие обучать и выполнять модель в браузере, используя JavaScript как среду выполнения.

Можно даже обучить модель TensorFlow на Python, а затем развернуть в среде JavaScript в браузере. Кроме того, если клиентское устройство оснащено графическим процессором (GPU), то Tensorflow.js сможет использовать его.

8.2.4. Плюсы и минусы

Основным преимуществом динамического развертывания на устройстве пользователя является то, что обращения к модели производятся быстро. Кроме того, снижается нагрузка на серверы организации, потому что большая часть вычисления выполняется на стороне пользователя. К тому же если модель развернута в браузере, то инфраструктура организации должна только отдавать веб-страницу, включающую параметры модели. Недостаток же развертывания в браузере в том, что может увеличиться стоимость трафика и время запуска приложения. Пользователь должен скачивать параметры модели при каждом запуске веб-приложения, а не один раз при установке приложения.

Еще один недостаток связан с обновлением модели. Напомним, что размер сериализованного объекта может быть весьма значителен. Некоторые пользователи могут оказаться не в сети в момент обновления или даже вообще отключить обновления. И тогда мы будем иметь пользователей, работающих с совершенно различными версиями моделей. А раз так, то возникают проблемы с обновлением серверной части приложения.

Развертывание модели на устройстве пользователя означает, что модель может быть подвергнута анализу со стороны конкурентов. Они могут попытаться выполнить обратную разработку модели и воспроизвести ее поведение. Или поискать слабые места, подавая на вход различные данные и наблюдая за выходом. Или адаптировать свои данные, так чтобы модель предсказывала то, что им нужно.

Предположим, что мобильное приложение позволяет пользователю читать интересные ему новости. Поставщик контента может подвергнуть модель обратной разработке и сделать так, что она будет чаще предлагать именно его новости.

Развертывание на устройстве пользователя, как и статическое, затрудняет мониторинг качества модели.

8.3. ДИНАМИЧЕСКОЕ РАЗВЕРТЫВАНИЕ НА СЕРВЕРЕ

Из-за перечисленных выше трудностей и проблем с мониторингом качества чаще всего модель развертывают на сервере (или нескольких серверах) и предоставляют доступ к ней как к веб-сервису с помощью **REST API** или как к службе удаленного вызова процедур **Google (gRPC)**.

8.3.1. Развертывание на виртуальной машине

В типичной архитектуре веб-сервиса, развернутого в облачной среде, предсказания отдаются в ответ на стандартно отформатированные HTTP-запросы. Веб-сервис, работающий на виртуальной машине, получает от пользователя запрос, содержащий входные данные, передает эти данные системе машинного обучения, после чего преобразует выход системы в строку в формате JSON или XML. Чтобы справиться с высокой нагрузкой, параллельно запускается несколько идентичных виртуальных машин.

Балансировщик нагрузки направляет входящие запросы той виртуальной машине, которая в данный момент наименее загружена. Виртуальные машины можно добавлять и удалять вручную или управлять ими как частью **автомасштабируемой группы**, которая запускает и останавливает виртуальные машины, исходя из текущей потребности. На рис. 8.2 показана такая схема развертывания. Каждый экземпляр, обозначенный оранжевым квадратиком, содержит весь код, необходимый для работы экстрактора признаков и модели. Экземпляр также содержит веб-сервис, имеющий доступ к этому коду.

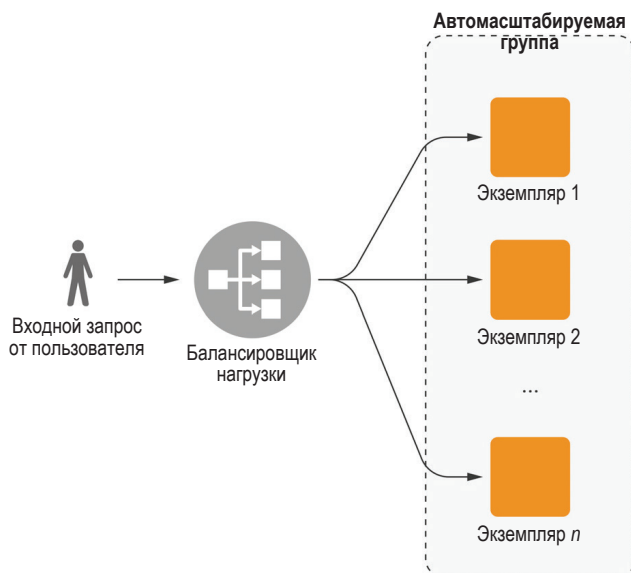


Рис. 8.2 ❖ Развертывание модели машинного обучения как веб-сервиса на виртуальной машине

В Python веб-сервис, основанный на REST API, обычно реализуется с помощью каркаса веб-приложений, например **Flask** или **FastAPI**. В R эквивалентом является **Plumber**.

В состав TensorFlow, популярного каркаса для обучения глубоких моделей, входит TensorFlow Serving – встроенная служба gRPC.

Преимущество развертывания на виртуальной машине – концептуальная простота архитектуры программной системы: это типичный веб-сервис или служба gRPC.

Из недостатков отметим необходимость обслуживать серверы (физические или виртуальные). При использовании виртуализации возникают дополнительные накладные расходы на виртуализацию и выполнение нескольких операционных систем. Еще один недостаток – сетевая задержка, это может стать серьезной проблемой, если требуется быстро обрабатывать результаты оценки. Наконец, развертывание на виртуальной машине может оказаться сравнительно дорогим по сравнению с развертыванием в контейнере или бессерверным развертыванием, которое мы обсудим ниже.

8.3.2. Развертывание в контейнере

Более современная альтернатива развертыванию на виртуальной машине – развертывание на основе контейнера. Считается, что работа с контейнерами более эффективна с точки зрения потребления ресурсов и обладает большей гибкостью, чем виртуальные машины. Контейнер похож на виртуальную машину в том смысле, что тоже является изолированной средой выполнения с собственной файловой системой, процессором, памятью и адресным про-

странством процесса. Основное отличие состоит в том, что все контейнеры работают на одной и той же виртуальной или физической машине и разделяют одну операционную систему, тогда как каждая виртуальная машина исполняет свой экземпляр ОС.

Процесс развертывания выглядит следующим образом. Система машинного обучения и веб-сервис устанавливаются в контейнер. Обычно используется **Docker**-контейнер, но есть и альтернативы. Затем система оркестрации организует выполнение контейнеров в кластере физических или виртуальных серверов. В качестве системы оркестрации для выполнения локально или в облаке чаще всего используют **Kubernetes**. Некоторые облачные платформы предоставляют свои движки оркестрации, например **AWS Fargate** и **Google Kubernetes Engine**, но при этом поддерживают также Kubernetes.

На рис. 8.3 показан этот паттерн развертывания. Здесь виртуальные или физические машины собраны в кластер, ресурсами которого управляет оркестратор контейнеров. В кластер можно вручную добавить новые виртуальные или физические машины или удалить из него старые. Если система развернута в облачной среде, то механизм автоматического масштабирования может запускать (и добавлять в кластер) или останавливать виртуальные машины, исходя из загрузки кластера.

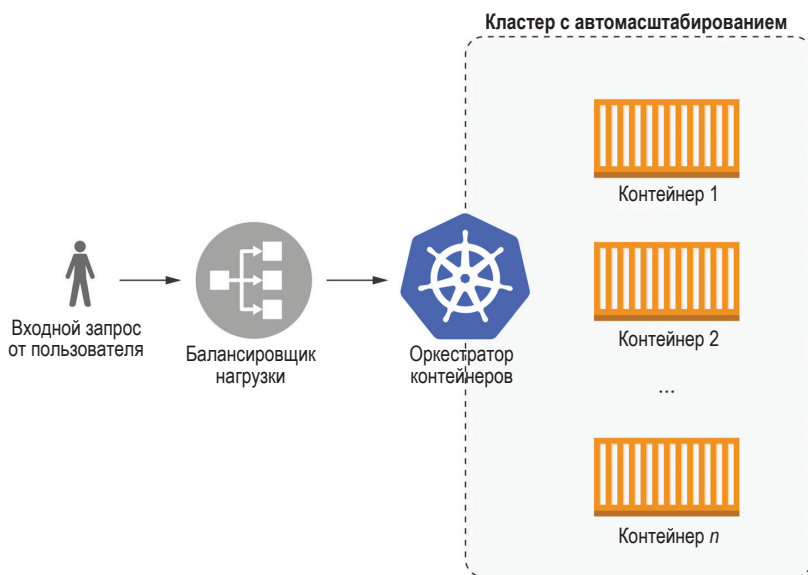


Рис. 8.3 ❖ Развертывание модели как веб-сервиса в контейнере, работающем в кластере

Развертывание в контейнере позволяет более эффективно использовать ресурсы по сравнению с развертыванием в виртуальной машине. Кроме того, есть возможность автоматического масштабирования в зависимости от интенсивности запросов. Допускается также **масштабирование до нуля**, когда количество контейнеров может быть снижено до нуля в случае простоя

и снова увеличено, когда появляются запросы. Таким образом, потребление ресурсов ниже, чем у постоянно работающих служб. Это снижает энергопотребление и позволяет сэкономить на оплате облачных ресурсов.

Недостаток развертывания в контейнере – сложность и необходимость привлечения квалифицированных кадров.

8.3.3. Бессерверное развертывание

Несколько поставщиков облачных услуг, в т. ч. Amazon, Google и Microsoft, предлагают так называемые **бессерверные вычисления**. У Amazon Web Services они называются Lambda-functions, а у Microsoft Azure и Google Cloud Platform – Functions.

Для бессерверного развертывания следует подготовить zip-архив, содержащий весь код, необходимый для выполнения системы машинного обучения (модель, экстрактор признаков и код вычисления оценки). В архиве должен быть файл с определенным именем, который содержит определенную функцию или метод класса с известной сигнатурой (точка входа). Архив загружается на облачную платформу и регистрируется под уникальным именем.

Облачная платформа предоставляет API для отправки входных данных бессерверной функции. Должны быть заданы ее имя и полезная нагрузка. В ответ возвращаются результаты. Облачная платформа берет на себя развертывание кода и модели, выделение адекватного количества ресурсов, выполнение кода и отправку результатов клиенту.

Обычно время выполнения функции, размер zip-файла и объем оперативной памяти, доступной во время выполнения, лимитируются поставщиком облачных услуг.

Лимит на размер файла может стать проблемой. Типичная модель машинного обучения требует нескольких тяжеловесных зависимостей. Для работы модели зачастую необходимы библиотеки на Python, в т. ч. Numpy, SciPy и scikit-learn. В зависимости от облачной платформы могут поддерживаться и другие языки программирования, в т. ч. Java, Go, PowerShell, Node.js, C# и Ruby.

У бессерверного развертывания есть много преимуществ. Очевидное – то, что не нужно подготавливать такие ресурсы, как серверы и виртуальные машины. Не нужно устанавливать зависимости, обслуживать и обновлять систему. Бессерверные системы масштабируются в широких пределах и могут без особых усилий поддерживать тысячи запросов в секунду. Бессерверные функции поддерживают синхронный и асинхронный режимы работы.

Кроме того, бессерверное развертывание экономически эффективно: вы платите только за время вычислений. Этой цели можно достичь и при двух других паттернах развертывания с помощью автомасштабирования, но для этого варианта характерна заметная задержка. Когда спрос снижается, лишние виртуальные машины могут продолжать работать, пока не будут остановлены.

Бессерверное развертывание также упрощает **канареечное развертывание**. В программной инженерии так называется стратегия, при которой обновленный код отправляется только небольшой группе конечных пользователей, обычно без их ведома. Поскольку новая версия распространяется малому числу пользователей, изменения можно быстро откатить, если об-

наружатся дефекты. Сконфигурировать две версии бессерверных функций в производственном режиме нетрудно, после чего можно отправлять одной из них умеренный трафик и протестировать ее, не тревожа много пользователей. Мы еще вернемся к канареечному развертыванию в разделе 8.4.

Откат при бессерверном развертывании тоже выполняется очень просто, потому что для переключения на предыдущую версию функции нужно всего лишь заменить один zip-архив.

Мы упомянули об ограничении на размер архива и объем памяти, доступной во время выполнения. Это важные недостатки бессерверного развертывания. Кроме того, серьезным ограничением при развертывании глубоких моделей может стать отсутствие доступа к GPU¹.

Конечно, в комплексных программных системах могут сочетаться разные паттерны развертывания. Паттерн, пригодный для одной модели, может оказаться неоптимальным для другой. Комбинирование нескольких паттернов развертывания называется **паттерном гибридного развертывания**. У персональных помощников типа Google Home или Amazon Echo может иметься модель, которая распознает фразу активации (например, «OK, Google» или «Алеха») и развертывается на клиентском устройстве, тогда как более сложные модели, умеющие обрабатывать запросы вида «помести песню X на устройство Y», работают на сервере. С другой стороны, система, развернутая на мобильном устройстве пользователя, могла бы дополнять видео и накладывать простые спецэффекты в режиме реального времени. Модель, развернутая на сервере, в таком случае использовалась бы для применения более сложных эффектов, например стабилизации и суперразрешения.

8.3.4. Потокоее развертывание модели

Паттерн **потокоее развертывания модели** можно рассматривать как нечто противоположное REST API. В интерфейсе REST API клиент отправляет запрос серверу, а затем ждет ответа (предсказания).

В комплексных системах к одному входу может применяться несколько моделей. Или же модель может получать на входе предсказание другой модели. Например, входом может быть новостная статья. Одна модель предсказывает тему статьи, другая извлекает именованные сущности, третья генерирует реферат и т. д.

Паттерн развертывания, основанный на REST API, предполагает существование отдельного API для каждой модели. Клиент должен был бы вызвать один API, отправив в составе запроса новостную статью, и получить в ответ ее тему. Затем клиент вызывает другой API, отправляя ему ту же статью, и получает именованные сущности. И так далее.

Потокоее развертывание работает иначе. Вместо того чтобы заводить по одному REST API на модель, все модели, а также необходимый для их работы код регистрируются в **движке потоковой обработки** (stream-processing engine – SPE). Примерами могут служить **Apache Storm**, **Apache Spark** и **Apache Flink**. Или же они упаковываются в приложение, основанное на **библиотеке**

¹ По состоянию на июль 2020 года.

поточковой обработки (stream-processing library – SPL), например **Apache Samza**, **Apache Kafka Streams** или **Akka Streams**.

Описания этих SPE и SPL выходят за рамки книги, но у всех них есть свойство, отличающее их от приложений на основе REST API. В каждом приложении потоковой обработки имеется явное или неявное понятие **топологии обработки данных**. Входные данные поступают в виде бесконечного потока элементов данных от клиента. Сообразуясь с предопределенной топологией, каждый элемент данных подвергается преобразованиям в узлах топологии. Преобразованный поток течет в другие узлы.

В потоковом приложении каждый узел как-то преобразует свой вход, а затем либо

- отправляет свой выход другим узлам, либо
- отправляет свой выход клиенту, либо
- сохраняет выход в базе данных или в файловой системе.

Один узел мог бы принимать новость и предсказывать ее тему, другой принимал бы новость и предсказанную тему и генерировал реферат и т. д.

Различие между приложением на основе REST API и потоковым приложением показано на рис. 8.4. Клиент, который пользуется REST API (рис. 8.4(a)), обрабатывает один элемент данных, например новостную статью, отправляя серию запросов. Различные REST API поочередно получают запросы и синхронно отправляют ответы. С другой стороны, потоковый клиент (рис. 8.4(b)) открывает соединение с потоковым приложением, отправляет запрос и получает события обновления по мере их возникновения.

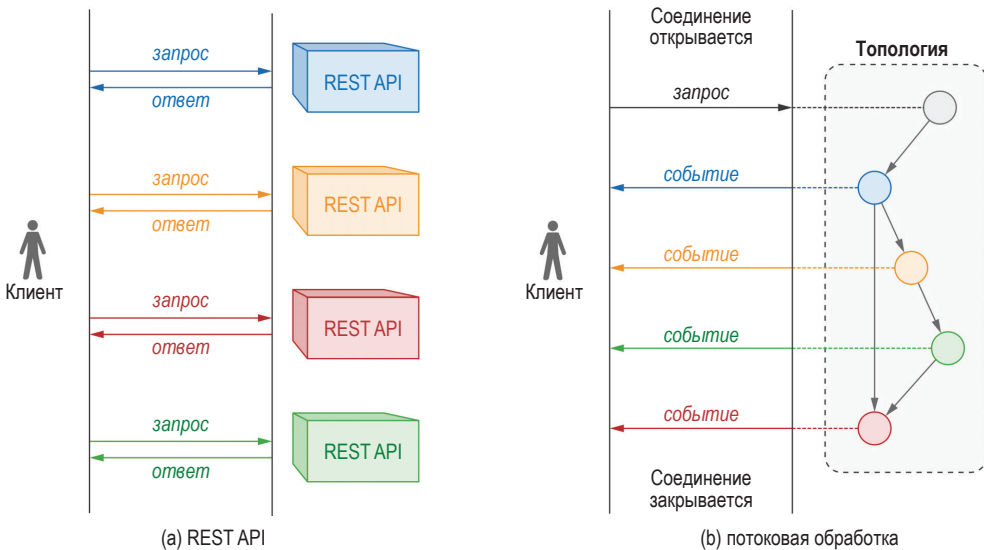


Рис. 8.4 ❖ Различие между REST API и потоковой обработкой:

- (a) чтобы обработать один элемент данных, клиент, пользующийся REST API, отправляет серию запросов, один за другим, и синхронно получает ответы;
- (b) чтобы обработать один элемент данных, потоковый клиент открывает соединение, отправляет запрос и получает события обновления по мере их возникновения

В правой части рис. 8.4(b) изображена топология, определяющая поток данных в приложении. Каждый входной элемент, отправленный клиентом, проходит через все узлы **топологического графа**. Его вершины могут посылать события обновления клиенту и (или) сохранять данные в базе или в файловой системе.

Потоковое приложение на основе SPE работает в собственном кластере виртуальных или физических машин и занимается распределением обработки данных между имеющимися ресурсами. Потоковому приложению на основе SPL для обработки данных не нужен выделенный кластер. Его можно интегрировать с имеющимися ресурсами, например виртуальными или физическими машинами либо с оркестратором контейнеров (типа Kubernetes).

REST API обычно применяются, чтобы дать клиентам возможность отправлять ситуативные запросы, для которых нет никакой часто повторяющейся закономерности. Это лучший выбор, когда клиент хочет сам решать, что делать с каждым ответом API. С другой стороны, если каждый запрос клиента

- типичен,
- подвергается определенной серии преобразований, особенно включающей несколько промежуточных, и
- всегда заканчивается одними и теми же действиями, например сохранением определенных элементов данных в файловой системе или в базе данных,

то потоковые приложения обеспечивают более эффективное использование ресурсов, меньшую задержку, безопасность и отказоустойчивость.

8.4. СТРАТЕГИИ РАЗВЕРТЫВАНИЯ

К типичным стратегиям развертывания относятся:

- разовое развертывание;
- немое развертывание;
- канареечное развертывание;
- многорукий бандит.

Рассмотрим их поочередно.

8.4.1. Разовое развертывание

Разовое развертывание самое простое. Концептуально это означает, что новая модель сериализуется в виде файла, а затем старый файл заменяется новым. Кроме того, если необходимо, заменяется экстрактор признаков.

Для развертывания на сервере в облачной среде мы подготавливаем новую виртуальную машину или контейнер, исполняющий новую версию модели. Затем заменяем образ виртуальной машины или контейнера. Наконец, мы постепенно закрываем старые машины или контейнеры и даем возможность механизму автомасштабирования запустить новые.

Для развертывания на физическом сервере мы загружаем файл новой модели (и при необходимости экстрактор признаков) на сервер. Затем заменяем старые файлы и код новыми версиями и перезапускаем веб-сервис.

Для развертывания на устройстве пользователя мы отправляем файл новой модели на устройство вместе с объектом выделения признаков и перезапускаем программу.

Если используется интерпретируемый код, то объект экстрактора признаков можно развернуть, заменив один файл с исходным кодом другим. Чтобы не развертывать повторно все приложение на сервере или на устройстве пользователя, объект экстрактора признаков можно сериализовать в файл. Затем при каждом запуске программа, исполняющая модель, будет десериализовывать объект.

Преимущество разового развертывания – простота. Но это и самая рискованная стратегия. Если новая модель или экстрактор признаков содержит ошибку, то последствия испытают все пользователи.

8.4.2. Немое развертывание

Немое развертывание дополняет разовое. В этом случае развертывается новая модель и новый экстрактор признаков, но старые сохраняются. Обе версии работают параллельно. Однако пользователям новая версия не предъявляется, пока не произойдет переключение. Предсказания, сделанные новой версией, только записываются в журнал. Спустя некоторое время они анализируются на предмет наличия ошибок.

Немое развертывание хорошо тем, что дает достаточно времени, чтобы удостовериться в правильности работы новой модели, не мешая пользователям. А недостаток – в том, что приходится выполнять в два раза больше моделей, для чего нужны дополнительные ресурсы. Кроме того, во многих приложениях просто невозможно оценить новую модель, не предъявляя ее предсказаний пользователям.

8.4.3. Канареечное развертывание

Напомним, что при **канареечном развертывании** новая версия модели и кода отправляется небольшой доле пользователей, тогда как большинство продолжает работать со старой моделью. В противоположность немому развертыванию такой подход позволяет контролировать качество новой модели и последствия ее предсказаний. В отличие от разового, возможные ошибки при канареечном развертывании не сказываются на всех пользователях.

Выбирая канареечное развертывание, мы принимаем дополнительную сложность, связанную с необходимостью одновременно обслуживать несколько версий модели.

Очевидный недостаток канареечного развертывания – невозможность обнаружить редкие ошибки. Если новая версия развернута у 5 % пользователей, а ошибка возникает в 2 % случаев, то шансов на ее обнаружение всего 0.1 %.

8.4.4. Многорукие бандиты

В разделе 7.3 мы видели, что **многорукие бандиты** (МAB) – это способ сравнения нескольких версий модели в производственной среде и выбора наилучшей. У МAB есть интересное свойство: после начального периода исследования, когда алгоритм МAB собирает факты с целью оценки качества каждой модели (рычага), наступает момент, начиная с которого все время «нажимается» лучший рычаг. Это означает, что колько скоро алгоритм сошелся, практически все пользователи направляются на лучшую версию модели.

Таким образом, алгоритм МAB одновременно решает две задачи – онлайн-новое оценивание модели и развертывание модели.

8.5. АВТОМАТИЗИРОВАННОЕ РАЗВЕРТЫВАНИЕ, ВЕРСИОНИРОВАНИЕ И МЕТАДААННЫЕ

Модель – это важный объект, но она никогда не поставляется сама по себе. Имеются дополнительные объекты для тестирования производственной модели, гарантирующие ее исправность.

8.5.1. Объекты, сопровождающие модель

Развертывайте модель в производственной среде, только если она сопровождается следующими объектами:

- **сквозной набор**, который определяет входы и выходы, для которых модель всегда должна работать;
- **доверительный тестовый набор**, который корректно определяет входы и выходы модели и используется для вычисления значения метрики;
- **метрика качества**, значения которой будут вычисляться на доверительном тестовом наборе путем применения к нему модели;
- **диапазон допустимых значений** метрики качества.

После того как система, использующая модель, установлена на сервере или клиентском устройстве, внешний процесс должен испытать модель на сквозных тестовых данных и убедиться, что все предсказания правильны. Затем тот же внешний процесс должен проверить, что значение метрики качества, вычисленное путем применения модели к доверительному тестовому набору, находится в допустимых пределах. Если хотя бы одна из этих проверок не проходит, то модель не должна обслуживать клиентов.

8.5.2. Синхронизация версий

Версии следующих трех элементов всегда должны быть синхронизированы:

- 1) обучающие данные;
- 2) экстрактор признаков;
- 3) модель.

При каждом обновлении данных должна быть создана новая версия в репозитории данных. Модель, обученная на конкретной версии данных, должна быть помещена в репозиторий моделей под таким же номером версии, как и у обучающих данных.

Даже если экстрактор признаков не изменялся, все равно его версия должна быть изменена и синхронизирована с версией модели и данных. Если же экстрактор был изменен, то необходимо построить новую модель с новым экстрактором и увеличить версии экстрактора, модели и обучающих данных (даже если последние не изменялись).

Развертывание новой модели должно быть автоматизированным и транзакционным. Получив номер версии подлежащей развертыванию модели, скрипт извлекает модель и экстрактор признаков из соответствующих репозиториях и копирует их в производственную среду. Модель должна быть применена к сквозным и доверительным тестовым данным путем имитации вызова извне. Если на сквозных данных возникает ошибка предсказания или на доверительных данных значение метрики качества выходит за пределы допустимого диапазона, то все развертывание следует откатить.

8.5.3. Метаданные версии модели

Каждая версия модели должна сопровождаться следующим кодом и метаданными:

- имя и версия библиотеки или пакета, использованного для обучения модели;
- если для построения модели применялся Python, то файл requirements.txt виртуальной среды, в которой создавалась модель (или имя Docker-образа, указывающее на путь в Docker Hub или в вашем реестре Docker);
- имя алгоритма обучения, имена и значения гиперпараметров;
- список признаков, необходимых модели;
- список выходов, их типы и описание того, как они должны потребляться;
- версия и местоположение данных, использованных для обучения модели;
- версия и местоположение контрольных данных, использованных для настройки гиперпараметров модели;
- код оценки, который применяет модель к новым данным и выводит предсказание.

Метаданные и код оценки должны быть сохранены в базе данных или в текстовом файле формата JSON либо XML.

Для аудита каждое развертывание модели должно также сопровождаться следующей информацией:

- кто и когда построил модель;
- кто, когда и на каком основании принял решение о развертывании этой модели;
- кто проводил экспертизу модели на предмет конфиденциальности и соответствия требованиям безопасности.

8.6. РЕКОМЕНДАЦИИ ПО РАЗВЕРТЫВАНИЮ МОДЕЛИ

В этом разделе мы обсудим практические аспекты развертывания систем машинного обучения в производственной среде. А также поделимся некоторыми полезными советами по развертыванию моделей.

8.6.1. Эффективность алгоритма

Аналитики данных чаще всего пишут на Python или R. Хотя существуют веб-каркасы, позволяющие создавать веб-сервисы на этих языках, сами языки не входят в число наиболее эффективных.

Большая часть кода научных пакетов на Python на самом деле написана на эффективном языке C или C++, а затем откомпилирована для конкретной операционной системы. Однако ваш код предобработки данных, извлечения признаков и оценки может быть не так эффективен.

Кроме того, не все алгоритмы практически полезны. Одни могут решить задачу быстро, другие работают слишком медленно. Для некоторых задач быстрых алгоритмов вообще не существует.

Целый раздел информатики – **анализ алгоритмов** – посвящен определению и сравнению сложности алгоритмов. Для классификации алгоритмов по скорости роста времени работы или потребляемой памяти при росте размера входных данных применяется **нотация O большое**.

Например, пусть требуется найти два примера в одномерном наборе данных S размера N , расстояние между которыми максимально. Вот возможный вариант алгоритма на Python для решения этой задачи:

```
1 def find_max_distance(S):
2     result = None
3     max_distance = 0
4     for x1 in S:
5         for x2 in S:
6             if abs(x1 - x2) >= max_distance:
7                 max_distance = abs(x1 - x2)
8                 result = (x1, x2)
9     return result
```

и эквивалентный ему на R:

```
1 find_max_distance <- function(S) {
2     result <- NULL
3     max_distance <- 0
```

```

4   for (x1 in S) {
5     for (x2 in S) {
6       if (abs(x1 - x2) >= max_distance) {
7         max_distance <- abs(x1 - x2)
8         result <- c(x1, x2)
9       }
10    }
11  }
12  result
13 }

```

В обоих случаях мы в цикле перебираем все элементы S и на каждой итерации внешнего цикла снова перебираем все элементы S . Следовательно, алгоритм выполняет N^2 сравнений чисел. Если считать, что время выполнения операций сравнения `abs` и присваивания равно 1, то временная сложность (или просто сложность) этого алгоритма составляет не менее $5N^2$. На каждой итерации производится одно сравнение, два вычисления `abs` и два присваивания ($1 + 2 + 2 = 5$). Для измерения сложности алгоритма в худшем случае используется нотация O большое. Конкретно для этого алгоритма сложность равна $O(N^2)$; постоянные типа 5 игнорируются.

Для решения той же задачи можно написать другой алгоритм на Python:

```

1  def find_max_distance(S):
2      result = None
3      min_x = float("inf")
4      max_x = float("-inf")
5      for x in S:
6          if x < min_x:
7              min_x = x
8          if x > max_x:
9              max_x = x
10     result = (max_x, min_x)
11     return result

```

или эквивалентный ему на R:

```

1  find_max_distance <- function(S):
2      result <- NULL
3      min_x <- Inf
4      max_x <- -Inf
5      for (x in S) {
6          if (x < min_x) {
7              min_x <- x
8          }
9          if (x > max_x) {
10             max_x = x
11         }
12     result <- c(max_x, min_x)
13     result

```

Теперь мы перебираем элементы S только один раз, поэтому сложность алгоритма составляет $O(N)$. В таком случае говорят, что второй алгоритм эффективнее первого.

Алгоритм называется **эффективным**, если его сложность полиномиально зависит от размера входных данных. Поэтому алгоритмы со сложностью $O(N)$ и $O(N^2)$ эффективны, т. к. N является полиномом первой степени, а N^2 – полиномом второй степени. Однако для очень больших N алгоритм со сложностью $O(N^2)$ будет все-таки слишком медленным. В эпоху больших данных ученые и инженеры стараются найти алгоритмы со сложностью $O(\log N)$.

При реализации алгоритма на практике следует по возможности избегать циклов, используя средства работы с векторами, имеющиеся в библиотеке NumPy и ей подобных. Например, в Python для вычисления скалярного произведения двух векторов $w \cdot x$ следует написать

```
1 import numpy
2 wx = numpy.dot(w,x)
```

а не

```
1 wx = 0
2 for i in range(N):
3     wx += w[i]*x[i]
```

Аналогично в R следует написать

```
1 wx = w %*% x
```

а не

```
1 wx <- 0
2 for (i in seq(N)):
3     wx <- wx + w[i]*x[i]
```

Используйте подходящие **структуры данных**. Если порядок элементов коллекции не играет роли, используйте множество вместо списка. В Python проверка принадлежности примера набору S производится быстро, если S – множество, но медленно, если S – список.

Еще одна важная структура данных в Python – dict. Она называется **словарем** или **хеш-таблицей** в некоторых других языках и позволяет определить коллекцию пар ключ–значение, в которой поиск ключа производится очень быстро.

Использование библиотек в общем случае надежнее. Писать свой код следует, только если вы занимаетесь научными исследованиями или когда без этого никак не обойтись. Python-пакеты для научных расчетов – NumPy, SciPy, scikit-learn и др. – написаны опытными учеными и инженерами, которые уделяли большое внимание эффективности. Многие методы в них написаны на C или C++ ради максимальной производительности.

Если требуется обойти большую коллекцию элементов, пользуйтесь **генераторами** в Python (а в R – их аналогами, находящимися в пакете **iterators**), которые создают функцию, возвращающую по одному элементу за раз, а не все сразу.

Используйте пакет **cProfile** в Python (аналог в R называется **lineprof**) для поиска неэффективных мест в коде.

Наконец, если алгоритм уже никак нельзя улучшить, то можно повысить быстродействие еще двумя способами:

- с помощью пакета **multiprocessing** в Python или **parallel** в R, который позволяет распараллеливать вычисления. Можно также воспользоваться каркасом распределенной обработки типа **Apache Spark**;
- с помощью **PyPy**, **Numba** или аналогичных инструментов для компиляции написанного на Python кода (в R имеется аналогичный пакет **compiler**) в быстрый оптимизированный машинный код.

8.6.2. Развертывание глубоких моделей

Иногда для достижения требуемой скорости необходимо вычислять оценку с помощью графического процессора (GPU). Плата за экземпляр GPU в облачной среде, как правило, гораздо выше, чем за «нормальный» экземпляр. Поэтому развертывать в облаке с одним или несколькими GPU следует только модели, оптимизированные для быстрого предсказания. Все остальные части приложения можно разместить отдельно в среде, где имеются только CPU. Такой подход позволяет снизить затраты, но при этом могут увеличиться накладные расходы на передачу данных между двумя частями приложения.

8.6.3. Кеширование

Кеширование – стандартная практика в программной инженерии. Кеш в памяти используется для хранения результатов вызова функции, чтобы при следующем вызове с такими же параметрами можно было прочитать результат из кеша.

Кеширование позволяет ускорить приложение, содержащее ресурсоемкие функции, которые долго вычисляются или часто вызываются с одними и теми же параметрами. В машинном обучении такими ресурсоемкими функциями являются модели, особенно те, что работают на GPU.

Простейший кеш можно реализовать в самом приложении. Например, на Python декоратор `lru_cache` позволяет обернуть функцию **запоминающим** вызываемым объектом, который сохраняет до `maxsize` последних результатов вызова:

```

1  from functools import lru_cache
2
3  # Прочитать модель из файла
4  model = pickle.load(open("model_file.pkl", "rb"))
5
6  @lru_cache(maxsize=500)
7  def run_model(input_example):
8      return model.predict(input_example)
9
10 # Теперь можно вызывать run_model
11 # с новыми данными

```


При первом вызове функции `run_model` с некоторыми входными параметрами будет вызван метод `model.predict`. При последующих вызовах `run_model` с такими же параметрами результат будет прочитан из кеша, запоминающего результаты `maxsize` последних обращений к `model.predict`.

В R аналогичного эффекта можно добиться с помощью функции `memo`:

```
1 library(memo)
2
3 model <- readRDS("./model_file.rds")
4
5 run_model <- function(input_example) {
6   result <- predict(model, input_example)
7   result
8 }
9
10 # Создать версию run_model с запоминанием
11 run_model_memo <- memo(run_model, cache = lru_cache(500))
12
13 # Теперь можно использовать run_model_memo
14 # вместо run_model для новых данных
```

Хотя использование `lru_cache` и других подобных подходов – большое удобство для аналитика, в крупных системах обычно применяются масштабируемые и конфигурируемые кеши общего назначения, например **Redis** или **Memcached**.

8.6.4. Формат доставки модели и кода

Напомним, что сериализация – самый простой способ доставить модель и код экстрактора признаков в производственную среду.

В любом современном языке программирования имеются инструменты сериализации. В Python это **Pickle**:

```
1 import pickle
2 from sklearn import svm, datasets
3
4 classifier = svm.SVC()
5 X, y = datasets.load_iris(return_X_y=True)
6 classifier.fit(X, y)
7
8 # Сохранить модель в файл
9 with open("model.pickle", "wb") as outfile:
10   pickle.dump(classifier, outfile)
11
12 # Прочитать модель из файла
13 classifier2 = None
14 with open("model.pickle", "rb") as infile:
15   classifier2 = pickle.load(infile)
16 if classifier2:
17   prediction = classifier2.predict(X[0:1])
```

А в R – RDS:

```

1 library("e1071")
2
3 classifier <- svm(Species ~ ., data = iris, kernel = 'linear')
4
5 # Сохранить модель в файле
6 saveRDS(classifier, "./model.rds")
7
8 # Прочитать модель из файла
9 classifier2 <- readRDS("./model.rds")
10
11 prediction <- predict(classifier2, iris[1,])

```

В scikit-learn лучше пользоваться заменой pickle – модулем **joblib**, который более эффективно работает с объектами, включающими большие массивы **NumPy**:

```

1 from joblib import dump, load
2
3 # Сохранить модель в файле
4 dump(classifier, "model.joblib")
5
6 # Прочитать модель из файла
7 classifier2 = load("model.joblib")

```

Такой же подход можно применить для сохранения сериализованного объекта экстрактора признаков в файле, копирования файла в производственную среду и последующего чтения из файла.

Для некоторых приложений скорость предсказаний критична. В таких случаях производственный код пишется на компилируемом языке, например Java или C/C++. Если аналитик построил модель на Python или R, то для развертывания в производственной среде есть три варианта:

- переписать код на компилируемом языке программирования;
- использовать какой-нибудь стандарт представления моделей, например PMML или PFA;
- использовать специализированный движок выполнения, например MLeap.

Язык разметки предсказывающих моделей (Predictive Model Markup Language – **PMML**) – это основанный на XML формат обмена предсказывающими моделями, который позволяет сохранять и передавать модели между совместимыми с PMML приложениями. PMML дает аналитику возможность разработать модель в приложении одного поставщика, а использовать в приложениях других поставщиков, так что проприетарные тонкости и несовместимости больше не являются препятствием для обмена моделями.

Например, допустим, что мы построили модель SVM на Python, а затем сохранили ее в PMML-файле. Предположим, что в производственной среде используется виртуальная машина Java (JVM). При условии что библиотека машинного обучения для JVM поддерживает формат PMML и в этой библиотеке имеется реализация SVM, нашу модель можно использовать в произ-

водственной среде без каких-либо изменений. Не надо ни переписывать код, ни заново обучать модель.

Переносимый формат для аналитики (Portable Format for Analytics – **PFA**) – еще один недавно появившийся стандарт для представления статистических моделей и систем преобразования данных. PFA позволяет переносить модели и конвейеры машинного обучения между гетерогенными системами и может похвастаться алгоритмической гибкостью. Модели, а также преобразования пред- и постобработки – это функции, которые можно произвольным образом компоновать, сцеплять или встраивать в сложные технологические процессы. Конфигурационный PFA-файл представляется в формате JSON или YAML.

Существуют общие «вычислители» с открытым исходным кодом для моделей или конвейеров, сохраняемые в виде PMML или PFA-файлов. Наиболее широко известны **JPMMML** (Java PMML) и **Hadrian**.

Вычислитель читает модель или конвейер из файла, исполняет ее, применяя к входным данным, и выводит предсказание.

К сожалению, PMML и PFA слабо поддерживаны популярными библиотеками и каркасами машинного обучения¹. Например, scikit-learn не поддерживает эти стандарты, хотя имеются сторонние проекты, например **SkLearn2PMML**, которые умеют преобразовывать объекты scikit-learn в формат PMML.

С другой стороны, такие движки, как **MLeap**, умеют быстро выполнять модели и конвейеры машинного обучения в среде JVM. На момент написания книги MLeap мог выполнять модели и конвейеры, созданные в Apache Spark и scikit-learn.

А теперь дадим еще несколько полезных практических советов касательно развертывания моделей.

8.6.5. Начинайте с простой модели

Развертывание и исполнение модели в производственной среде может быть сложнее, чем кажется. После того как инфраструктура для обслуживания простой модели устаканилась, можно переходить к обучению и развертыванию более сложных моделей.

Простую интерпретируемую модель проще отлаживать, особенно с точки зрения экстракторов признаков и всего конвейера машинного обучения. У сложных моделей и конвейеров много зависимостей и настраиваемых гиперпараметров, они более уязвимы к ошибкам реализации и развертывания.

8.6.6. Тестируйте на посторонних

Прежде чем передавать модель в эксплуатацию, протестируйте ее на посторонних, а не только на тестовых данных. Посторонними могут быть другие члены команды или сотрудники компании. Можно также прибегнуть

¹ По состоянию на июль 2020 года.

к краудсорсингу или привлечь реальных пользователей, согласившихся принять участие в экспериментах с новыми возможностями продукта.

Тестирование на посторонних поможет избежать необъективности, поскольку создатель модели всегда эмоционально привязан к ней. Это также позволит предъявить модель пользователям другого типа (например, если в вашей команде есть только мужчины или выходцы с Кавказа).

8.7. РЕЗЮМЕ

Есть несколько паттернов развертывания модели: статически, как часть устанавливаемого пакета, динамически на устройстве пользователя, динамически на сервере или потоком.

У статического развертывания много преимуществ, например быстрое выполнение, ненарушенная конфиденциальность пользователя и возможность обращаться к модели без подключения к сети. Есть и недостаток: труднее обновить модель без обновления всего приложения в целом.

Главное преимущество динамического развертывания на устройствах пользователей – быстрое обращение к модели. Кроме того, уменьшается нагрузка на серверы организации. Недостатки – трудность доставки обновлений всем пользователям и доступность модели для стороннего анализа.

Как и статическое развертывание, развертывание на устройстве пользователя затрудняет мониторинг качества модели.

Динамическое развертывание на сервере может принимать одну из следующих форм: развертывание на виртуальной машине, развертывание в контейнере и бессерверное развертывание.

Самый популярный способ – развернуть модель на сервере и сделать ее доступной с помощью REST API, раскрываемого веб-сервисом или службой gRPC. В этом случае клиент отправляет запрос серверу, а затем ждет ответа, прежде чем отправить следующий запрос.

Потоковое развертывание модели устроено иначе. Все модели регистрируются в движке потоковой обработки или упаковываются в виде приложения, основанного на библиотеке потоковой обработки. Клиент отправляет один запрос и получает обновления по мере их возникновения.

Типичные стратегии развертывания: разовое, немое, канареечное и многорукий бандит.

При разовом развертывании новая модель сериализуется в виде файла, затем старый файл заменяется новым.

Немое развертывание подразумевает развертывание старой и новой версий и их параллельную работу. Новая версия не предъявляется пользователю, пока не произойдет переключение. Предсказания новой версии только записываются в журнал и анализируются. Поэтому остается достаточно времени, чтобы удостовериться в правильной работе новой модели, не беспокоя пользователей. Недостаток в том, что приходится выполнять больше моделей, расходуя дополнительные ресурсы.

Канареечное развертывание заключается в том, что новая версия отправляется небольшой доле пользователей, а большинство продолжает работать

со старой версией. Это позволяет проверить качество модели и оценить ее удобство для пользователей. В случае ошибки пострадает не так много пользователей.

Многорукие бандиты позволяют развернуть новую модель, оставив старую. Алгоритм заменяет старую модель новой, только когда уверен, что новая работает лучше.

Развертывание новой версии модели должно быть автоматизированным и транзакционным. Получив номер версии подлежащей развертыванию модели, скрипт извлекает модель и экстрактор признаков из соответствующих репозиторий и копирует их в производственную среду. Модель должна быть применена к сквозным и доверительным тестовым данным путем имитации вызова извне. Если на сквозных данных возникает ошибка предсказания или на доверительных данных значение метрики качества выходит за пределы допустимого диапазона, то все развертывание следует откатить.

Версии обучающих данных, экстрактора признаков и модели должны быть синхронизированы.

Эффективность алгоритмов – важный аспект развертывания модели. Python-пакеты для научных расчетов – NumPy, SciPy, scikit-learn и др. – написаны опытными учеными и инженерами, которые уделяли большое внимание эффективности. Ваш собственный код может оказаться не таким эффективным и надежным. Писать свой код следует, только если это абсолютно необходимо.

При реализации собственного алгоритма избегайте циклов. Используйте подходящие структуры данных. Если порядок элементов коллекции не играет роли, используйте множество вместо списка. Использование словарей (или хеш-таблиц) позволяет определить коллекцию пар ключ–значение, в которой поиск ключа производится очень быстро.

Кеширование ускоряет приложение, если оно содержит ресурсоемкие функции, которые часто вызываются с одними и теми же параметрами. В машинном обучении такими ресурсоемкими функциями являются модели, особенно исполняемые на GPU.

Глава 9

Выполнение, мониторинг и сопровождение модели

В этой главе мы рассмотрим передовые практики выполнения, мониторинга и сопровождения модели в производственной среде. Это последние три этапа жизненного цикла проекта машинного обучения.

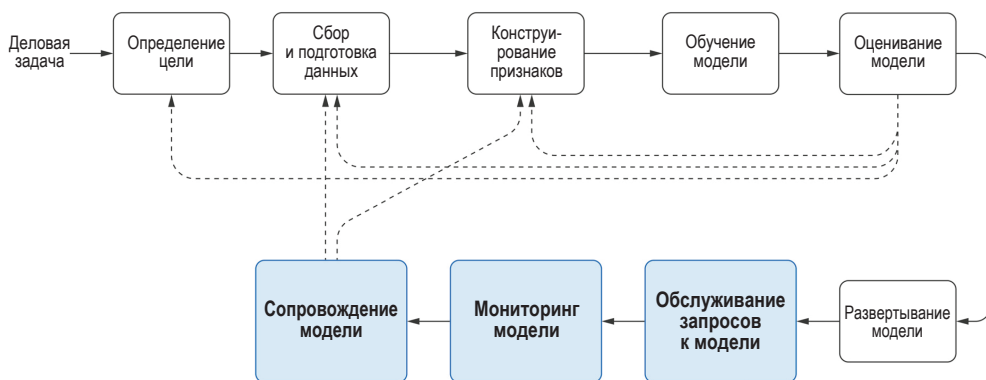


Рис. 9.1 ❖ Жизненный цикл проекта машинного обучения

В частности, мы охарактеризуем свойства среды выполнения модели машинного обучения, в которой модель применяется к входным данным, и режимы выполнения модели – пакетный и по запросу. Кроме того, мы рассмотрим три основные проблемы выполнения модели в реальном мире: ошибки, изменения и человеческая природа. Мы опишем, за чем именно следует наблюдать в производственной среде и как и когда обновлять модель.

9.1. Свойства среды выполнения модели

Среда выполнения модели – это то место, где модель применяется к входным данным. Ее свойства определяются **паттерном развертывания** модели. Однако у эффективной среды выполнения могут быть дополнительные свойства, которые мы здесь и обсудим.

9.1.1. Безопасность и корректность

Среда выполнения отвечает за аутентификацию пользователя и авторизацию его запросов.

Вот что необходимо проверять:

- разрешен ли пользователю доступ к модели, которую он хочет выполнить;
- соответствуют ли спецификации модели имена и значения переданных параметров;
- доступны ли пользователю эти параметры и их значения.

9.1.2. Простота развертывания

Среда выполнения должна допускать обновление модели с минимальными усилиями, в идеале не требуя обновлять приложение целиком. Если модель была развернута как веб-сервис на физическом сервере, то для обновления должно быть достаточно заменить один файл модели другим и перезапустить веб-сервис.

Если модель была развернута как экземпляр виртуальной машины или контейнер, то экземпляры или контейнеры, исполняющие старую версию модели, должны заменяться путем постепенной остановки работающих и запуска новых из нового образа. Тот же принцип применим к оркестрированным контейнерам.

Модель с потоковым развертыванием обычно обновляется путем передачи потока, содержащего новую версию. Для этого потоковое приложение должно хранить информацию о состоянии. После того как новая версия и связанные с ней компоненты (например, экстрактор признаков и код предсказания) переданы приложению, состояние приложения изменяется и теперь включает новую версию этих объектов. Современные **движки потоковой обработки** поддерживают приложения с состоянием. Описанная архитектура схематически изображена на рис. 9.2.

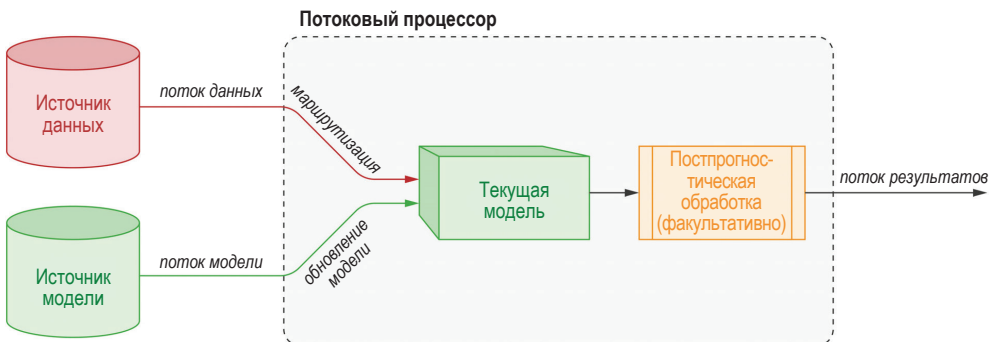


Рис. 9.2 ❖ Общая архитектура потокового развертывания модели

9.1.3. Гарантии правильности модели

Эффективная среда выполнения автоматически проверяет, что исполняемая ей модель правильна. Кроме того, она проверяет, что модель, экстрактор признаков и другие компоненты синхронизированы. Это следует проверять при каждом запуске веб-сервиса или потокового приложения и периодически во время работы. В разделе 8.5 мы говорили, что каждую развернутую модель должны сопровождать следующие четыре объекта: **сквозной набор, доверительный тестовый набор, метрика качества и диапазон допустимых значений**.

Модель не должна запускаться в производственной среде (а если запущена, то должна быть немедленно остановлена), если выполняется хотя бы одно из двух условий:

- по меньшей мере один из сквозных тестовых примеров оценен неправильно;
- значение метрики на примерах из доверительного тестового набора оказалось вне допустимого диапазона.

9.1.4. Простота восстановления

Эффективная среда выполнения допускает восстановление после ошибок простым откатом к предыдущей версии.

Восстановление после неудачного развертывания должно производиться так же и с такой же легкостью, как развертывание обновленной модели. Единственное отличие – вместо новой модели развертывается предыдущая работающая версия.

9.1.5. Предотвращение расхождений между обучением и выполнением

Настоятельно рекомендуется избегать создания двух разных кодовых баз: одной для обучения модели, другой для предсказаний в производственной среде. Что касается **выделения признаков**, то малейшее расхождение между двумя версиями экстрактора может стать причиной неоптимальной, а то и вообще неправильной работы модели.

Инженерная группа может повторно реализовать экстрактор признаков для производственной среды по многим причинам. Самая распространенная – что код, написанный аналитиком данных, неэффективен или несовместим с производственной экосистемой.

Таким образом, среда выполнения должна допускать простой доступ к коду выделения признаков для различных надобностей, в т. ч. для повторного обучения модели, ситуативных обращений к модели и работы в производственной среде. Один из способов решить эту задачу – обернуть объект выделения признаков отдельным веб-сервисом.

Если никак невозможно избежать использования двух разных кодовых баз для генерирования признаков в режиме обучения и в режиме эксплуатации, то среда выполнения должна хотя бы обеспечить протоколирование значений признаков, сгенерированных в производственной среде. Эти значения затем должны использоваться как обучающие.

9.1.6. Предотвращение скрытых петель обратной связи

В разделе 4.12 главы 4 мы видели пример **скрытой петли обратной связи**. Модель m_B использовала выход модели m_A в качестве признака, не зная, что модель m_A , в свою очередь, использовала в качестве признака выход модели m_B .

В другом примере скрытой петли обратной связи участвует только одна модель. Предположим, что имеется модель, которая классифицирует входные почтовые сообщения как спам или не спам. Допустим, что пользовательский интерфейс позволяет пользователю помечать сообщения как спам или не спам. Очевидно, что мы хотели бы использовать помеченные сообщения для улучшения модели. Однако, поступая так, мы рискуем создать скрытую петлю обратной связи – и вот почему.

В нашем приложении пользователь помечает сообщение как спам, когда видит его. Однако пользователь видит только те сообщения, которые наша модель классифицировала как не спам. Кроме того, маловероятно, что пользователь будет регулярно заходить в папку спама и помечать некоторые сообщения как не спам. Следовательно, на действия пользователя оказывает существенное влияние наша модель, из-за чего получаемые от пользователя данные смещены: мы воздействуем на явление, на котором хотим обучаться.

Чтобы избежать смещения, помечайте небольшой процент примеров как «зарезервированные» и показывайте их пользователю еще до применения модели. И только эти зарезервированные примеры используйте для дополнительного обучения, даже те из них, на которые пользователь не отреагировал.

В более общем случае одна модель может косвенно влиять на данные, используемые для обучения другой модели. Допустим, что одна модель решает, в каком порядке отображать книги, а другая – какие отзывы показывать рядом с каждой книгой. Если первая модель помещает отзыв на некоторую книгу в конец списка, то отсутствие реакции пользователя на отзыв, показанный второй моделью, может быть вызвано его положением в списке, а не качеством самого отзыва.

9.2. Режимы выполнения модели

Модели машинного обучения могут выполняться в пакетном режиме или по запросу. Ответы на запросы могут быть предназначены человеку или машине.

9.2.1. Выполнение в пакетном режиме

В пакетном режиме модель обычно выполняется, когда применяется к большим объемам входных данных. Например, когда модель используется для полной обработки данных всех пользователей продукта или службы. Или когда она систематически применяется ко всем входным событиям, например твитам или комментариям к онлайн-новостям. В пакетном режиме ресурсы используются более эффективно, чем по запросу, а применяется он, когда некоторая задержка допустима.

При выполнении в пакетном режиме модель обычно принимает от ста до тысячи векторов признаков за раз. Поэкспериментируйте, чтобы найти оптимальный с точки зрения быстродействия размер пакета. Как правило, выбираются размеры, являющиеся степенями двойки: 32, 64, 128 и т. д.

Результаты обработки пакета обычно сохраняются в базе данных, а не отправляются конкретным потребителям. Пакетный режим можно использовать, например, чтобы:

- построить список еженедельных рекомендаций новых песен всем пользователям музыкального потокового сервиса;
- классифицировать поток комментариев к новостям и статьям в блогах как спам или не спам;
- извлечь именованные сущности из документов, проиндексированных поисковой системой, и т. д.

9.2.2. Обслуживание запроса со стороны человека

Обслуживание запроса к модели со стороны человека подразумевает выполнение шести шагов:

- 1) проверка запроса;
- 2) получение контекста;
- 3) преобразование контекста во вход модели;
- 4) применение модели к входу и получение выхода;
- 5) проверка осмысленности выхода;
- 6) представление выхода пользователю.

Прежде чем модель будет обслуживать запрос, поступивший от пользователя, возможно, придется проверить, есть ли у пользователя права на доступ к этой модели.

Контекст представляет ситуацию, в которой пользователь отправил запрос к системе машинного обучения и получил от нее ответ.

Пользователь может отправить запрос явно или неявно. Пример явного запроса – когда пользователь музыкального потокового сервиса запрашивает рекомендации к песням, похожим на данную. А неявный запрос отправляет мессенджер, запрашивая рекомендуемые ответы на самое недавнее сообщение, полученное пользователем.

Хороший контекст можно собирать в режиме реального или почти реального времени. Он будет содержать информацию, необходимую экстрактору признаков для генерирования значений всех признаков, ожидаемых мо-

делью. Он также содержит информацию, которая будет использована для улучшения модели со временем.

Рассмотрим примеры хорошего контекста для нескольких задач.

Устройство плохо работает

Если обнаружено неправильное функционирование устройства, то хороший контекст должен включать уровни вибрации и шума, информацию о задаче, решаемой устройством, идентификатор пользователя, версию прошивки, время с момента сбоя и с момента последнего техобслуживания, количество использований с момента изготовления и с момента последнего техобслуживания.

Перевод в палату интенсивной терапии

Для решения о том, нужно ли поместить нового пациента в палату интенсивной терапии, хороший контекст должен включать возраст, кровяное давление, температуру, частоту сердцебиений, показатели пульсоксиметрии, общий анализ крови, биохимический анализ крови, газовый анализ артериальной крови, уровень алкоголя в крови, медицинский анамнез и наличие беременности.

Оценка риска невозврата кредита

Для принятия решения об одобрении или отказе в выдаче кредитной карты хороший контекст должен включать возраст, образование, сведения о занятости, статус резидента страны, годовой доход, семейное положение, невыплаченные долги, наличие других кредитных карт, является ли заявитель собственником или арендатором жилплощади, объявлял ли он о личном банкротстве, пропускал ли он в прошлом выплаты по кредиту и если да, то сколько раз. Даже если какая-то информация не нужна для выделения признаков, она все равно может быть необходима для протоколирования и отладки, например идентификатор клиента, дата и время суток.

Показ рекламного объявления

Для принятия решения о том, следует ли показать некоторое рекламное объявление, хороший контекст должен включать название веб-страницы, положение пользователя на странице, разрешение экрана, текст на странице и текст, видимый пользователю, сведения о том, как пользователь пришел на эту страницу и сколько провел на ней времени. Для протоколирования и отладки контекст может включать версию браузера, версию операционной системы, информацию о подключении, дату и время.

Экстрактор признаков преобразует контекст во входные данные модели. Иногда экстрактор является частью **конвейера** машинного обучения (см. раздел 5.4). Но чаще он создается как отдельный объект.

Когда результат оценки должен быть отправлен человеку, он редко представляется в неизменном виде. Обычно предсказание модели преобразуется в форму, допускающую более простую интерпретацию, и именно она передается клиенту.

Прежде чем отправлять предсказание человеку, обычно вычисляют для него оценку доверия. Если доверие низкое, то, возможно, не стоит сообщать пользователю ничего: пользователи меньше жалуются на ошибки, которых не видели. Или если пользователь все-таки ожидает получить что-то, проинформируйте его о низкой достоверности предсказания и задайте вопрос «Вы уверены?».

Такого рода вопросы особенно важны, когда система может инициировать некоторое действие на основе предсказания. Если вы можете оценить возможную цену ошибки и если достоверность предсказания принадлежит диапазону (0, 1), то умножьте величину $(1 - \text{достоверность})$ на цену, чтобы получить оценку стоимости неверного действия. Например, пусть цена ошибки оценивается в 1000 долларов и модель дает оценку достоверности предсказанию 0.95, тогда **ожидаемая стоимость ошибки** равна $(1 - 0.95) \times 1000 = 50$ долларов. Можно задать порог ожидаемой стоимости для разных действий, рекомендуемых моделью, и спрашивать пользователя, если ожидаемая стоимость превышает порог.

Помимо вычисления достоверности предсказания, проверяйте, имеет ли предсказание смысл. В разделе 9.3 мы подробнее опишем, что проверять и какой должна быть реакция системы, если результат не имеет смысла.

Удобно сохранять в журнале контекст, в котором модель выдавала предсказания, и реакцию пользователя. Это поможет впоследствии найти причины проблем и улучшить модель, добавив обучающие примеры.

9.2.3. Обслуживание запроса со стороны машины

При построении REST API, рассчитанного на разные случаи, мы часто сталкиваемся с необходимостью обслуживать запросы от машины потоком. На самом деле машины обычно предъявляют стандартные и предопределенные требования к данным. Хорошо продуманная фиксированная топология потокового приложения позволяет эффективно использовать имеющиеся ресурсы.

Обслуживание по запросу (не важно, со стороны машины или человека) может представлять трудности. Интенсивность запросов может меняться от очень высокой днем до очень низкой ночью. Если используются виртуальные ресурсы в облаке, то может помочь **автомасштабирование**, когда ресурсы добавляются по мере возрастания спроса и освобождаются, когда спрос снижается. Но автомасштабирование недостаточно проворно, чтобы справиться со случайными всплесками спроса.

Для таких ситуаций, когда спрос сильно варьирует, разработаны архитектуры **брокеров сообщений**, например **RabbitMQ** или **Apache Kafka**. Брокер позволяет одному процессу записывать сообщения в очередь, а другому читать из этой очереди. Запросы помещаются во входную очередь. Среда выполнения модели периодически подключается к брокеру. Она читает порцию входных элементов данных из входной очереди и генерирует предсказания для каждого элемента в пакетном режиме. Затем предсказания помещаются в выходную очередь. Другой процесс периодически подключается к брокеру,

читает предсказания из выходной очереди и отправляет их пользователям, приславшим запросы (рис. 9.3). Такая архитектура не только позволяет справиться с пиковым спросом, но и более эффективно использует ресурсы.

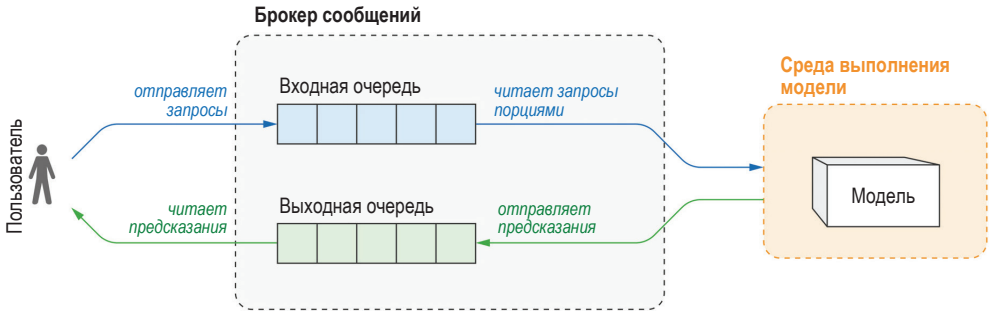


Рис. 9.3 ❖ Модель обслуживания по запросу с помощью брокера сообщений

9.3. ВЫПОЛНЕНИЕ МОДЕЛИ НА ПРАКТИКЕ

На практике при обслуживании запросов со стороны реальных людей модель сталкивается с трудностями. Обычно невозможно предсказать все действия и реакции пользователей. Архитектура программной системы, предназначенной для работы в реальном мире, должна предусматривать три фактора: ошибки, изменение и человеческую природу.

9.3.1. Готовность к ошибкам

В любой программной системе ошибки неизбежны. В системах на основе машинного обучения ошибки являются неотъемлемой частью решения: совершенных моделей не бывает. Поскольку мы не можем исправить все ошибки, остается только научиться с ними жить.

Это означает, что программную систему нужно проектировать так, чтобы при возникновении ошибки она продолжала нормально функционировать.

Существует три «не», которые мы должны принять и включить в систему:

1. Мы не всегда можем объяснить, почему произошла ошибка.
2. Мы не можем надежно предсказать, когда ошибка возникнет, и даже предсказание с высокой достоверностью может оказаться неправильным.
3. Мы не всегда знаем, как исправить конкретную ошибку. Если ее вообще можно исправить, то сколько и каких обучающих данных для этого необходимо?

Кроме того, когда ошибка случается, мы не всегда вправе ожидать, что неправильное предсказание хотя бы близко к правильному или похоже на него. Ошибка может быть сколь угодно «безумной». Например, модель беспилотного автомобиля при скорости 120 км/ч и полном отсутствии препятствий может предсказать, что наилучшее действие – остановиться и сдать назад.

Малейшие изменения в контексте могут приводить к совершенно неожиданным и ошибочным результатам. Например, модель, которая распознает опасные ситуации в заводском цеху, может начать делать ошибки после замены электрической лампочки рядом с камерой. Раньше стояла лампа накаливания, а теперь – люминесцентная.

Даже редкие ошибки могут затронуть пользователей, если их много. Допустим, что верность модели составляет 99 %. Если пользователей миллион, то один процент ошибок предсказания затронет тысячи людей.

Редко случается, что исправление одной ошибки в модели приводит к новым ошибкам. Но гарантий нет.

Как проектировать систему, в которой ошибки неизбежны?

9.3.2. Отношение к ошибкам

Прежде всего следует определить стратегию, по крайней мере частично сглаживающую последствия предсказаний, в результате которых система выглядит или действует «глупо». Например, если система поддерживает диалог с пользователем, таким как персональный помощник или чат-бот, то лучше сказать «я не знаю», чем ляпнуть что-то наугад. Если пользователь видит ошибку, то вычислите **ожидаемую стоимость ошибки**, как было описано выше, и не показывайте предсказание, если эта стоимость выше порога.

Можно поступить по-другому – обучить вторую модель m_B , которая предсказывает, что первая модель m_A может допустить ошибку на данном входе. Наличие «охранной» модели m_B особенно важно, если модель m_A является частью особо ответственной системы.

Видимость ошибки – важный фактор при решении о том, следует ли ее скрывать, и если да, то как. Например, рассмотрим систему, которая скачивает веб-страницы из интернета и выделяет из них некоторые сущности. Допустим, пользователь хочет получать уведомления, когда обнаружена определенная сущность. Модель может допускать ошибки двух видов: 1) выделить сущность, хотя в документе ее нет (ложноположительный результат, FP), 2) не выделить сущность, присутствующую в документе (ложноотрицательный результат, FN). В первом случае пользователь получит ненужное уведомление и будет недоволен. Во втором случае пользователь не получит уведомления, ничего не узнает об ошибке и избежит отрицательных эмоций. В такой ситуации, возможно, стоит оптимизировать **точность** модели, сохранив **полноту** на разумно высоком уровне.

При обучении модели решите, какого рода ошибок вы хотели бы избежать в первую очередь, а затем настройте гиперпараметры, в том числе порог точности, соответственно.

Если уверенность в лучшем предсказании низкая, рассмотрите возможность представления нескольких вариантов. Именно поэтому Google предлагает сразу 10 результатов поиска. Шансы, что самая релевантная ссылка окажется среди этих 10 результатов, гораздо выше, чем шансы оказаться в первой позиции.

Другой способ избежать недовольства пользователя ошибками модели – дозировать взаимодействие пользователя с моделью. Измерьте, сколько

ошибок допускает модель, и оцените, сколько ошибок в минуту (день, неделю, месяц) пользователь готов вытерпеть. После этого ограничьте взаимодействия модели с пользователем, так чтобы количество наблюдаемых ошибок было меньше этого уровня.

Для тех ситуаций, когда ошибка произошла и пользователь мог ее увидеть, предусмотрите возможность отправить отчет об ошибке. Получив отчет, сохраните в журнале контекст, в котором использовалась модель, и ее предсказание. Объясните пользователю, какие действия будут предприняты, чтобы предотвратить появление подобных ошибок в будущем.

Было бы уместно измерять активность пользователей в системе, протоколировать все взаимодействия, а затем анализировать подозрительные в пакетном режиме. Вот что имеет смысл анализировать:

- взаимодействует ли пользователь с системой меньше, чем раньше;
- игнорировал ли пользователь некоторые рекомендации;
- проводил ли пользователь адекватное время в различных ситуациях.

Чтобы еще уменьшить негативное влияние ошибок, предоставьте пользователю возможность отменить рекомендованное системой действие, если система допускает нечто подобное. По возможности распространите этот режим на все автоматизированные действия, выполняемые системой от имени пользователя.

Приложения, действующие от имени пользователя, должны быть особенно ограничены в своих возможностях. Напомним, что ошибки моделей машинного обучения могут быть сколь угодно «безумными» – см. пример беспилотного автомобиля, решившего сдать назад. Осторожность нужно соблюдать и в других критических ситуациях, касающихся здоровья, безопасности или финансов, например участия в аукционах или выписывании рецептов на лекарства. Если модель предсказывает, что нужно купить или продать больше ценных бумаг, чем скользящее среднее плюс одно стандартное отклонение, то было бы разумно отправить уведомление и приостановить «автоматические» действия. Той же логики следует придерживаться, если модель предсказывает дать пациенту необычно высокую дозу лекарства или изменить скорость автомобиля на величину, значительно большую или меньшую, чем обычно.

Если ваша система может автоматически отвергать предсказание модели, то стоит реализовать какую-то резервную стратегию в дополнение к информированию пользователя об отказе (рис. 9.4). В качестве замены можно использовать менее изощренную модель или специальную эвристику. Конечно, выход резервной стратегии тоже должен проверяться и может быть отвергнут, если выглядит неразумно. В таком случае пользователю следует отправить сообщение об ошибке.

9.3.3. Готовность к изменениям и отношение к ним

Качество системы, основанной на машинном обучении, обычно изменяется со временем. В некоторых приложениях оно может изменяться в режиме почти реального времени.

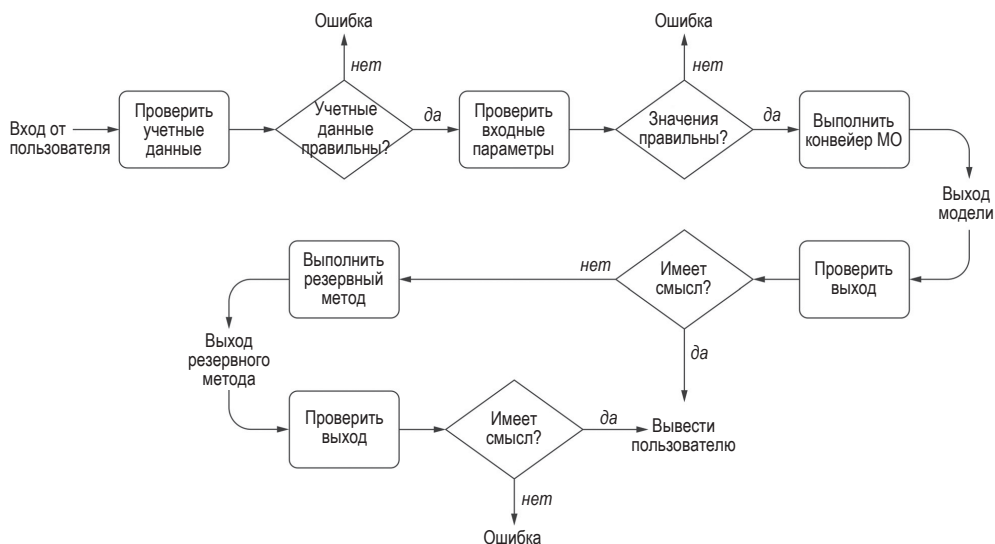


Рис. 9.4 ❖ Блок-схема выполнения модели на практике

Есть два типа изменения модели:

- 1) ее качество может стать лучше или хуже;
- 2) для некоторых входов она может давать другие предсказания.

Типичная причина снижения качества модели со временем – **смена концепта**, которую мы уже рассматривали в разделе 3.2.6. Представление о том, что считать правильным предсказанием, может измениться в связи со сменой предпочтений и интересов пользователя. Это могло бы потребовать повторного обучения модели на недавно помеченных данных.

Некоторые изменения могут восприниматься пользователями позитивно. Но иногда изменения воспринимаются негативно, даже если с инженерной точки зрения качество повысилось. Возможно, мы добавили обучающие примеры, заново обучили модель и наблюдаем улучшенное значение метрики качества. Однако при добавлении данных мы неосознанно внесли в данные несбалансированность. Теперь некоторые классы представлены недостаточно. Пользователи, которым интересны именно эти классы, видят снижение качества предсказаний и пишут жалобы или даже вовсе отказываются от системы.

Пользователи привыкают к определенному поведению. Они знают, какой запрос задать поисковой системе, чтобы получить часто используемый документ или веб-приложение. Возможно, этот запрос и не оптимален для данной цели, однако он работал. Но вы улучшили алгоритм ранжирования результатов по релевантности. И теперь система не возвращает некий документ или приложение или помещает его на вторую страницу результатов поиска. Пользователь больше не может найти документ, который раньше находил так легко, и, естественно, недоволен.

Если вы ожидаете, что пользователи могут негативно отреагировать на изменение, дайте им время привыкнуть. Расскажите пользователям об из-

менениях и о том, чего ожидать от новой модели. Или вносите изменения постепенно. Возможно, стоит смешивать предсказания старой и новой моделей и медленно уменьшать долю старой. Или можно выполнять старую и новую модели параллельно и дать пользователю возможность переключиться на старую модель, прежде чем окончательно похоронить ее.

9.3.4. Готовность к особенностям человеческой природы и отношение к ним

Человеческая природа – вот что стоит на пути эффективной системной инженерии. Люди непредсказуемы, зачастую ведут себя иррационально, непоследовательно, и чего они ожидают, не понятно. Хорошая программная система должна все это учитывать.

Избегайте путаницы

Система должна быть спроектирована так, чтобы пользователь не приходил в замешательство при работе с ней. Выход модели следует представлять в интуитивно понятном виде, не предполагая, что пользователь что-то знает о машинном обучении и искусственном интеллекте. На самом деле многие пользователи будут считать, что работают с типичной программой, и появление ошибок вызовет у них недоумение.

Умерьте ожидания

С другой стороны, некоторые пользователи ждут от системы многого. Основная причина этого – реклама. Для привлечения внимания продукт или система, основанные на машинном обучении, часто подаются как «интеллектуальные». Например, персональные помощники типа Apple Siri, Google Home и Amazon Alexa нередко рекламируются так, будто наделены разумом. И правда, любая система на основе машинного обучения может выглядеть очень умной, если входные данные тщательно подобраны. Пользователи видят такую рекламу и экстраполируют ее на ситуации, в которых система не будет работать эффективно, потому что не была на них рассчитана.

Еще одна причина, по которой пользователи ожидают чего-то сверхъестественного (хотя им никто не обещал), – то, что они работали с похожей (в их понимании) системой, которая выглядела «очень интеллектуальной». И ждут такого же уровня от вашей системы.

Завоевывайте доверие

Некоторые пользователи, особенно опытные, вообще не доверяют системе, если знают, что она содержит какой-то «интеллект». Основная причина недоверия – прошлый опыт. Большинство так называемых интеллектуальных систем не оправдали ожиданий, поэтому некоторые пользователи внутренне готовы к неудаче, впервые столкнувшись с вашей системой.

Поэтому ваша система должна завоевывать доверие каждого пользователя, и как можно скорее. Пользователь, имеющий опыт работы с «интеллектуальными» системами, скорее всего, подвергнет вашу систему нескольким простым тестам. Если система их не пройдет, пользователь не будет ей доверять. Например, если вы разработали поисковую систему, то пользователь может ввести свое имя или созданный им документ и посмотреть, найдет ли его система. Или если система предоставляет интеллектуальные справки об организациях корпоративным клиентам, то пользователь проверит, сколько система знает о его организации и есть ли в ее «интеллекте» хоть толика смысла. Водитель беспилотного автомобиля, скорее всего, проверит такие команды, как «запусти двигатель», «следуй за той машиной», «двигайся с текущей скоростью» или «припаркуйся на той улице». В зависимости от природы службы следует предвидеть такие простые тесты и позаботиться о том, чтобы система их прошла.

Не переутомляйте пользователя

Усталость пользователя – еще одна причина, по которой он может утратить интерес к вашей системе. Следите, чтобы система не слишком часто прерывала работу пользователя своими рекомендациями или запросами на одобрение. Не вываливайте все, что у вас есть, разом. По возможности, дайте пользователю возможность выразить свою заинтересованность явно.

Добавим: из того, что система может автоматически выполнить некоторое действие, еще не следует, что она так и должна поступить. Например, система, которая автоматизирует взаимодействие пользователя с другими людьми, могла бы отправлять закрытые данные или данные для служебного пользования в ответе на почтовое сообщение или выкладывать их в открытый форум. Прежде чем раскрывать данные от имени пользователя, следует оценить, насколько информация конфиденциальна. Используйте модель, чтобы обнаруживать такие потенциально конфиденциальные тексты и изображения. Другая крайность – когда система чересчур осторожна и автоматически отфильтровывает релевантную информацию либо утомляет пользователя слишком частыми просьбами подтвердить решения.

Остерегайтесь фактора отторжения

При взаимодействии пользователя с системой на основе машинного обучения есть такое явление, как **фактор отторжения**. Это значит, что пользователь считает предсказательную способность модели слишком высокой. Пользователь ощущает дискомфорт, особенно если предсказание касается очень личных вещей. Следите за тем, чтобы система не воспринималась как «Большой брат» и не брала на себя слишком много.

9.4. Мониторинг модели

Развернутую модель нужно постоянно мониторить. Это помогает быть уверенными, что:

- результаты запросов к модели правильно доводятся до адресатов и
- качество модели не опускается ниже установленного предела.

9.4.1. Что может пойти не так?

Мониторинг призван вовремя предупреждать о проблемах с работой модели в производственной среде. К ним относятся:

- новые обучающие данные, использованные для обновления модели, сделали только хуже;
- реальные данные в производственной среде изменились, а модель нет;
- код выделения признаков претерпел существенную модификацию, но модель не адаптировалась к нему;
- ресурс, необходимый для генерирования признака, изменился или стал недоступен;
- модель используется во зло или подверглась атаке злоумышленников.

Дополнительные обучающие данные – не всегда хорошо. **Разметчик** мог неправильно интерпретировать инструкции по разметке. Или решения, принятые двумя разными разметчиками, противоречивы. Данные, автоматически собираемые для улучшения модели, могут быть смещены. Возможные причины: **скрытая петля обратной связи** (раздел 9.1.6) или **систематическая ошибка** (раздел 3.2).

Иногда свойства данных в производственной среде изменяются постепенно, но модель не адаптируется. Она по-прежнему основана на старых данных, которые уже не репрезентативны. Одна из возможных причин – **смена концепта**, которую мы обсуждали в разделе 9.3.

Программист мог исправить дефект в коде выделения признаков и обновить экстрактор признаков в производственной среде. Но если он при этом не обновил также саму модель, то качество может измениться непредсказуемо.

Даже если экстрактор признаков и модель синхронизированы, исчезновение или изменение некоторого ресурса (подключения к базе данных, таблицы базы данных или внешнего API) может повлиять на некоторые признаки, сгенерированные экстрактором.

Некоторые модели, особенно развернутые на платформах электронной торговли и медийных, становятся мишенями атак со стороны злоумышленников. Недобросовестные конкуренты, мошенники, преступники и другие государства могут активно искать слабые места модели и соответственно корректировать атаки. Если ваша система обучается на действиях пользователей, то некоторые из них могут специально действовать так, чтобы изменить поведение модели в свою пользу.

Кроме того, злоумышленники могут попытаться исследовать обученную модель, с тем чтобы получить информацию о ее обучающих данных. Эти данные потенциально могут содержать конфиденциальные сведения о людях и организациях.

Еще одна форма злонамеренного использования, предотвратить которую особенно трудно, – **двойное применение**. Как и любое программное обеспечение, модель машинного обучения можно использовать во благо (как мы и намеревались) или во зло (часто без нашего согласия). Например, мы могли

создать и выложить в открытый доступ модель, которая заставляет ваш голос звучать как у мультяшного персонажа. А мошенники могут адаптировать ваш результат, чтобы подделать голос клиента банка и произвести по телефону какую-то операцию от его имени. Или мы могли создать модель, распознающую пешеходов на улице. А производитель автоматического оружия воспользуется ей, чтобы распознавать людей на поле боя.

9.4.2. Что и как мониторить

Цель мониторинга – помочь нам обеспечить разумное качество модели на **доверительном тестовом наборе**. Этот набор следует регулярно обновлять с помощью новых данных, чтобы избежать **сдвига распределения**. Кроме того, модель нужно регулярно тестировать на примерах из **сквозного набора**.

Очевидно, что подходящими метриками для мониторинга являются верность, точность и полнота, но есть одна метрика, особенно полезная для измерения изменений со временем: **смещение предсказания**.

В статичном мире, где ничего не меняется, распределение предсказанных классов приблизительно совпадало бы с распределением наблюдаемых классов, особенно если модель **хорошо откалибрована**. Если на деле наблюдается противное, значит, налицо смещение предсказания. Это могло бы означать, что распределение меток обучающих данных и текущее распределение классов в производственной среде разошлись. Необходимо исследовать причины этого изменения и внести необходимые коррективы.

Мониторинг позволяет нам узнавать об исчезнувших или перепрофилированных источниках данных. В некоторые столбцы таблицы базы данных может прекратиться поступление информации. Определение или формат данных в некоторых столбцах может измениться, а не ведающая об этом модель будет предполагать, что все осталось по-прежнему. Чтобы избежать всего этого, необходимо мониторить распределение значений каждого признака, извлеченного из таблицы базы данных, на предмет сдвига. Сдвиг распределения как признаков, так и предсказаний можно обнаружить, применив статистические критерии, например **критерий независимости хи-квадрат** или **критерий Колмогорова–Смирнова**. При обнаружении значительного сдвига распределения следует отправить уведомление всем заинтересованным сторонам.

Следует также следить за **численной устойчивостью** модели и отправлять уведомление, если обнаружены значения NaN (нечисло) или бесконечность.

Важно наблюдать за вычислительной производительностью системы машинного обучения. Как мгновенное, так и медленное падение необходимо обнаруживать и рассылать предупреждения.

Следите также за подозрительными флуктуациями в использовании модели, в частности:

- отслеживайте количество запросов к модели в час и сравнивайте его с соответствующим значением за прошлый день. Посылайте преду-

прежде, если число изменилось на 30 % и более. Этот порог следует настроить с учетом условий в вашем приложении, чтобы избежать лишних предупреждений;

- отслеживайте количество запросов к модели за день и сравнивайте его с соответствующим значением за прошлую неделю. Посылайте предупреждение, если число изменилось на 15 % и более. Настройте порог под себя.

Мониторинг этих величин поможет обнаружить следующие нежелательные изменения:

- минимальных и максимальных предсказанных значений;
- значений медианы, среднего и стандартного отклонения предсказаний в течение заданного промежутка времени;
- задержки при обращении к API модели;
- потребления памяти и процессора при выполнении предсказаний.

Дополнительно, чтобы предотвратить сдвиг распределения, система автоматического мониторинга должна:

- 1) накапливать входные данные, случайным образом откладывая их в сторонку на протяжении некоторого периода времени;
- 2) отправлять эти данные для разметки;
- 3) выполнять модель и вычислять значения метрики качества;
- 4) уведомлять заинтересованные стороны о значительном снижении качества.

Рекомендательные системы нуждаются в дополнительном мониторинге. Эти модели предлагают рекомендации пользователям сайта или приложения. Бывает полезно мониторить кликабельность (CTR), т. е. отношение числа пользователей, щелкнувших по рекомендации, к общему числу пользователей, получивших рекомендацию от модели. Если CTR уменьшается, то модель следует обновить.

Важно обратить внимание на компромисс между чрезмерно консервативным поведением и частым уведомлением заинтересованных сторон об изменившейся метрике. Если вы поднимаете тревогу слишком часто, то люди устанут от постоянных уведомлений и начнут игнорировать их. Если система не критическая, то лучше позволить заинтересованным сторонам самостоятельно определить пороги отправки уведомлений.

Протоколируйте события мониторинга, чтобы процесс можно было проследить от начала до конца. Для визуального анализа качества модели пользовательский интерфейс средства мониторинга должен показывать диаграммы трендов, из которых видно, как модель деградирует со временем.

Среди прочего средство мониторинга должно уметь вычислять и визуализировать метрики на срезах данных. Срезом называется подмножество данных, включающее только такие примеры, для которых некий атрибут имеет заданное значение. Например, один срез может содержать только примеры, относящиеся к штату Флорида, другой – примеры, касающиеся женщин, и т. д. Не исключено, что деградация модели наблюдается лишь в некоторых срезах, а в остальных все нормально.

Помимо мониторинга в реальном времени, важно также протоколировать данные, которые:

- могут помочь в поиске источника проблемы;
- невозможно проанализировать в режиме реального времени;
- могут быть полезны для улучшения имеющихся моделей или обучения новых.

9.4.3. Что протоколировать

Важно протоколировать достаточно информации, чтобы можно было воспроизвести любое странное поведение в процессе будущего анализа. Если модель обслуживает пользователя какого-то графического интерфейса, например посетителя веб-сайта или пользователя мобильного приложения, то имеет смысл сохранять **контекст** пользователя в момент обращения к модели. Как было сказано в разделе 9.2, контекст может включать: содержимое веб-страницы (или состояние приложения), положение пользователя на странице, время суток, местонахождение пользователя и по каким ссылкам он переходил перед обращением к модели.

Кроме того, полезно включать входные данные модели, т. е. признаки, выделенные из контекста, и время, затраченное на генерирование этих признаков.

Журнал мог бы также включать:

- выход модели и время, затраченное на его генерирование;
- новый контекст пользователя после ознакомления с выходом модели;
- реакцию пользователя на выход модели.

Под реакцией пользователя понимается действие, последовавшее сразу после ознакомления с выходом модели: на что он нажал и через сколько времени после ознакомления с выходом.

В крупных системах с тысячами пользователей, когда каждый из них обращается к модели сотни раз в день, протоколировать каждое событие может оказаться невозможным. На практике разумнее производить **стратифицированную выборку**. Сначала нужно решить, какие группы событий мы хотим протоколировать, а затем протоколировать только определенный процент событий в каждой группе. Группы могут образовывать пользователи или контексты. Пользователей можно группировать по возрасту, полу или длительности использования сервиса (новые и давно зарегистрированные клиенты). Контексты можно группировать по времени суток: раннее утро, рабочие часы, поздний вечер.

Если данные о действиях пользователей хранятся в журналах, то пользователи должны знать, что, когда и как сохраняется и в течение какого срока хранится. По возможности данные должны быть анонимизированы или агрегированы без потери полезности. Доступ к конфиденциальным данным следует предоставлять только лицам, которым поручено решить конкретную задачу, и на определенный период времени. Не позволяйте аналитику работать с конфиденциальными данными для решения задач, не относящихся к деятельности организации. Это может быть незаконно.

У пользователей должна быть возможность явно запретить протоколирование и анализ данных об их действиях. В разных странах применяются раз-

ные политики хранения данных. В каждой стране имеются свои ограничения на то, какую информацию о ее гражданах можно хранить и использовать для анализа, а какую нельзя.

9.4.4. Мониторинг неправомерного использования

Некоторые лица или организации могут использовать вашу модель для собственных целей. Такие пользователи могут отправлять миллионы запросов в день, тогда как типичный пользователь отправляет только десяток. С другой стороны, у кого-то может возникнуть желание подвергнуть обучающие данные обратной разработке, чтобы заставить модель выдать желаемый результат.

Есть разные способы предотвратить неправомерное использование, например:

- взимать плату за каждый запрос;
- постепенно увеличивать паузы перед отправкой ответа на запрос и даже
- блокировать некоторых пользователей.

Для достижения своих целей некоторые злоумышленники могут попытаться манипулировать вашей моделью. Злоумышленник может подать на вход данные, которые изменят модель способом, выгодным только ему. В результате общее качество модели может пострадать.

Из способов противостоять таким действиям отметим следующие:

- не доверять данным от некоторого пользователя, если похожие данные не поступают от многих пользователей;
- назначить каждому пользователю репутационную оценку и не доверять данным, полученным от пользователей с низкой репутацией;
- классифицировать поведение пользователя как нормальное или аномальное и не принимать данные от пользователей с аномальным поведением.

Злоумышленники будут стараться обойти ваши защитные меры, изменив свое поведение. Чтобы эффективно защитить систему, регулярно обновляйте модели. Добавляйте новые данные и признаки, чтобы вовремя обнаружить мошеннические операции.

9.5. СОПРОВОЖДЕНИЕ МОДЕЛИ

Большинство эксплуатируемых моделей следует регулярно обновлять. Частота зависит от нескольких факторов:

- насколько часто система допускает ошибки и насколько они критичны;
- насколько актуальной должна быть модель, чтобы оставаться полезной;
- как быстро становятся доступны новые обучающие данные;
- сколько времени занимает повторное обучение модели;

- сколько стоит развертывание модели;
- каков вклад обновления модели в ценность продукта и достижение целей пользователей.

В этом разделе мы поговорим о сопровождении модели: когда и как обновлять модель после развертывания в производственной среде.

9.5.1. Когда обновлять

Когда модель развертывается в производственной среде впервые, она часто бывает далека от совершенства. Модель допускает ошибки предсказания – и это неизбежно. Некоторые ошибки могут быть критичны, поэтому модель нуждается в обновлении. Со временем модель «становится на ноги» и требует меньше обновлений. Однако некоторые модели нужно обновлять постоянно, чтобы они всегда оставались актуальными.

Актуальность модели зависит от потребностей бизнеса и пользователя. Рекомендательную модель на сайте интернет-магазина нужно обновлять после каждой покупки. Если пользователь прислушивается к рекомендациям модели, выбирая статьи на новостном сайте, то модель разумно обновлять раз в неделю. С другой стороны, модель распознавания и синтеза речи или модель машинного перевода можно изменять не так часто.

Тема появления новых обучающих данных также влияет на частоту обновления модели. Даже если данные поступают быстро, как в случае потока комментариев на популярном сайте, для их разметки может потребоваться время и значительные инвестиции. Иногда разметка автоматизирована, но производится с задержкой, как в **предсказании оттока клиентов**, когда решение пользователя продолжать пользоваться службой или уйти будет принято в далеком будущем.

Для обучения некоторых моделей требуется много времени, особенно если необходим **поиск гиперпараметров**. Не редкость – ждать несколько дней или даже недель. Чтобы ускорить обучение, пользуйтесь алгоритмами машинного обучения, допускающими распараллеливание, а также графическими процессорами (GPU). Современные библиотеки, например **thundersvm** и **cuML**, позволяют запускать алгоритмы поверхностного обучения на GPU и тем самым получать значительный выигрыш во времени. Если вы не можете себе позволить ждать получения обновленной модели несколько дней или недель, то пользуйтесь менее сложной (а значит, менее точной) моделью – другого выхода нет.

Если обновление обходится дорого, то, быть может, стоит обновлять модель не так часто. Например, в медицине получить помеченные примеры трудно и дорого в силу нормативных требований, соображений конфиденциальности и необходимости привлекать высокооплачиваемых специалистов.

Не все модели заслуживают развертывания. Иногда потенциальный выигрыш не окупается возможным недовольством пользователей. Однако если неудобства, доставляемые пользователям, можно смягчить, а развертывание стоит не очень дорого, то даже небольшое улучшение может в длительной перспективе принести весомые дивиденды бизнесу.

9.5.2. Как обновлять

Выше мы обсуждали, что в идеале программное обеспечение должно допускать развертывание новой версии модели без остановки всей системы. Инфраструктура на основе виртуальных машин (VM) или контейнеров позволяет сделать это следующим образом: заменить образ VM или контейнера в репозитории, постепенно закрыть работающие VM/контейнеры и позволить средству автомасштабирования создать новые VM/контейнеры из обновленного образа.

На рис. 9.5 схематически изображена архитектура автоматизации развертывания и сопровождения системы машинного обучения. Мы видим три репозитория: данных, кода и моделей; все три версионированы. Также имеется две среды выполнения: для обучения модели и производственная. Модель работает в производственной среде, к которой применяются балансировка нагрузки и автомасштабирование. Когда требуется обновить модель, среда построения модели извлекает обучающие данные и код обучения соответственно из репозитория данных и кода. Затем она обучает новую модель и сохраняет ее в репозитории моделей.

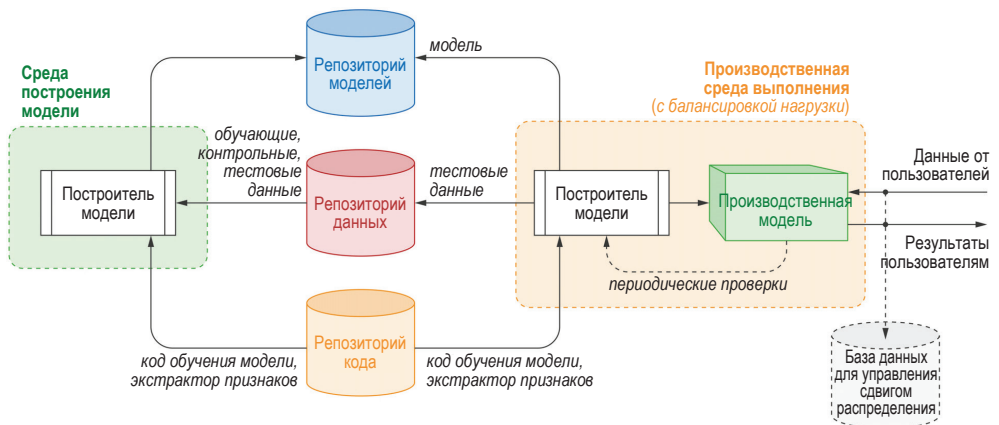


Рис. 9.5 ❖ Архитектура автоматизации развертывания и сопровождения системы машинного обучения

После того как новая версия модели оказалась в репозитории, производственная среда выполнения извлекает:

- новую модель из репозитория моделей;
- тестовые данные из репозитория данных;
- код, применяющий модель к тестовым данным, из репозитория кода.

Если новая модель проходит тесты, то старая модель изымается из производственной среды и заменяется новой с помощью подходящей стратегии развертывания, как обсуждалось в разделе 8.4. При принятии решения о замене могут помочь **А/В-тестирование** или алгоритм **многорукотого бандита**.

В базе данных для управления **сдвигом распределения** накапливаются входные данные, полученные моделью, и результаты их оценки. Когда будет накоплено достаточно данных, они отправляются для проверки человеку¹ с целью выявить сдвиг распределения.

В случае **потокowego развертывания модели** обновление производится, когда изменяется состояние потокового процессора (см. раздел 9.1.2 и рис. 9.2).

Если **модель выполняется по запросу** с применением архитектуры **брокера сообщений**, то ее обновление устроено примерно так же, как при потоковом развертывании (см. раздел 9.2.3 и рис. 9.3).

На рис. 9.6 показана архитектура на основе брокера сообщений, которая позволяет не только выполнять и обновлять модель, но и включает в себя **разметчика-человека**. Разметчик получает непомеченные примеры, производит из них выборку, назначает метки выбранным примерам и отправляет их обратно брокеру сообщений. Модуль обучения модели читает помеченные примеры из очереди. Когда их наберется достаточно для значимого обновления модели, он обучает новую модель, сохраняет ее в репозитории моделей и отправляет сообщение «модель готова» брокеру. Процесс обслуживания запросов к модели извлекает новую версию модели из репозитория и отбрасывает текущую модель.

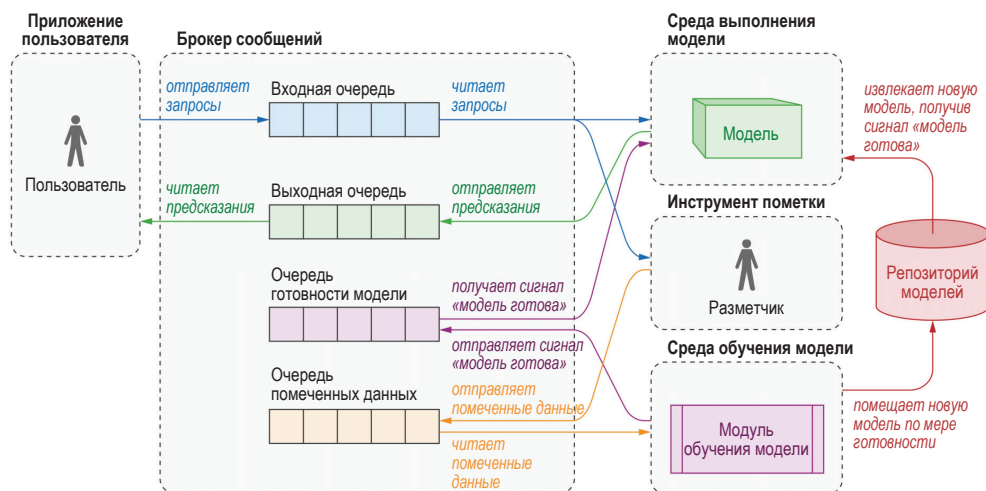


Рис. 9.6 ❖ Обновление модели и обслуживание запросов с применением брокера сообщений

Обсудим еще несколько соображений, касающихся сопровождения модели. Во многих компаниях применяется технологический процесс непрерывной интеграции, когда модель обучается автоматически, как только стано-

¹ Или автоматизированной системе, более точной, чем модель, которую нельзя разместить в производственной среде (например, потому что она слишком хрупкая, дорогая или медленная).

вятся доступны новые обучающие данные. Рекомендуется заново обучать модель с нуля, используя все обучающие данные, а не дообучать имеющуюся модель на одних только новых примерах.

Для каждого обучающего примера рекомендуется хранить идентификатор разметчика. Кроме того, связывайте с каждым значением в производственной базе данных версию модели, которая использовалась для генерирования этого значения. Если обнаружится проблема в этой версии, то информация о том, какие значения были ей сгенерированы, позволяет повторить обработку этих и только этих значений.

Если модель часто переобучается, то удобно хранить гиперпараметры конвейера в системе конфигурирования. Google дает следующие рекомендации¹ по организации хорошей системы конфигурирования:

1. Должно быть легко задать новую конфигурацию, как дельту по сравнению с предыдущей.
2. Должно быть трудно допустить ручные ошибки, пропуски или недосмотры.
3. Должно быть легко увидеть (глазами) различия между конфигурациями двух моделей.
4. Должно быть легко автоматически формулировать и проверять базовые утверждения о конфигурации: число используемых признаков, зависимости между данными и т. д.
5. Должна быть возможность обнаруживать неиспользуемые или избыточные настройки.
6. Конфигурации должны проходить полный цикл технического рецензирования и сохраняться в репозитории.

Убедитесь, что в среде выполнения достаточно места на жестком диске и оперативной памяти для размещения обновленной модели. Не ждите, что старая и новая версии модели будут различаться только качеством. Будьте готовы к ситуации, когда новая модель окажется гораздо больше предыдущей. И не ждите, что новая модель будет работать так же быстро, как предыдущая. Неэффективный код выделения признаков, дополнительная ступень конвейера или выбор другого алгоритма могут существенно повлиять на скорость предсказания.

Модели неизбежно делают ошибки при предсказании. Однако для компании или ее клиента одни ошибки могут обходиться дороже других. После развертывания новой версии модели проверяйте, что количество дорогих ошибок если и увеличилось, то не сильно.

Проверяйте, что ошибки равномерно распределены между всеми категориями пользователей. Нежелательно, чтобы новая модель работала хуже для пользователей из некоторой миноритарной группы или из конкретного места.

Если какая-то из описанных выше проверок не проходит, развертывать новую модель не рекомендуется. Откатитесь к предыдущей версии, если ошибка обнаружена после развертывания, и начните расследование. В разделе 9.1 мы говорили, что откат к предыдущей версии должен быть не сложнее развертывания новой.

¹ Sculley et al. Hidden Technical Debt in Machine Learning Systems (2015).

Остерегайтесь **каскадирования моделей**. В разделе 6.7.6 отмечалось, что если выход одной модели подается на вход другой, то изменение одной модели может повлечь за собой снижение качества другой. Если в системе используется каскадирование, то обновляйте сразу все модели, составляющие каскад.

9.6. РЕЗЮМЕ

Эффективная среда выполнения обладает следующими свойствами. Она безопасна и корректна, обеспечивает легкость развертывания и восстановления и предоставляет гарантии правильности модели. Кроме того, она избегает расхождений между обучением и выполнением и скрытых петель обратной связи.

Модели машинного обучения выполняются в пакетном режиме или по запросу. Запросы могут поступать от человека или от машины. Обычно модель выполняется в пакетном режиме, если применяется к большим данным и некоторая задержка допустима.

При выполнении запросов от человека модель обычно обернута REST API. Требования машины к данным обычно стандартные и предопределенные, поэтому часто такие запросы обслуживаются потоком.

Архитектура программной системы, предназначенной для работы в реальном мире, должна предусматривать три фактора: ошибки, изменение и человеческую природу.

Модель, развернутая в производственной среде, должна подвергаться постоянному мониторингу. Цели мониторинга: гарантировать, что модель отвечает на запросы правильно и что ее качество остается в допустимых пределах.

В производственной среде возможны различные негативные сценарии, например:

- новые обучающие данные, использованные для обновления модели, сделали только хуже;
- реальные данные в производственной среде изменились, а модель нет;
- код выделения признаков претерпел существенную модификацию, но модель не адаптировалась к нему;
- ресурс, необходимый для генерирования признака, изменился или стал недоступен;
- модель используется во зло или подверглась атаке злоумышленников.

Средства автоматизации должны вычислять значения метрик качества, критичных для бизнеса, и отправлять заинтересованным сторонам уведомления, если эти значения существенно изменились или оказались ниже порога. Кроме того, мониторинг должен обнаруживать сдвиг распределения, численную неустойчивость и снижение вычислительной эффективности.

Важно протоколировать достаточно информации, чтобы можно было воспроизвести любое странное поведение в процессе будущего анализа. Если модель обслуживает пользователя какого-то графического интерфейса, на-

пример посетителя веб-сайта или пользователя мобильного приложения, то имеет смысл сохранять контекст пользователя в момент обращения к модели. Кроме того, полезно включать входные данные модели, т. е. признаки, выделенные из контекста, и время, затраченное на генерирование этих признаков. Журнал может также включать выход модели и время, затраченное на его генерирование, новый контекст пользователя после ознакомления с выходом модели и реакцию пользователя на выход модели.

Некоторые пользователи могут использовать вашу модель для собственных целей. Они могут подвергнуть обучающие данные обратной разработке, чтобы заставить модель выдать желаемый результат. Чтобы предотвратить неправомерное использование, можно предпринять следующие меры:

- не доверять данным от некоторого пользователя, если похожие данные не поступают от многих пользователей;
- назначить каждому пользователю репутационную оценку и не доверять данным, полученным от пользователей с низкой репутацией;
- классифицировать поведение пользователя как нормальное или аномальное и не принимать данные от пользователей с аномальным поведением;
- взимать плату за каждый запрос;
- постепенно увеличивать паузы перед отправкой ответа на запрос;
- блокировать некоторых пользователей.

Большинство моделей машинного обучения необходимо обновлять – регулярно или от случая к случаю. Частота обновлений зависит от нескольких факторов:

- насколько часто система допускает ошибки и насколько они критичны;
- насколько актуальной должна быть модель, чтобы оставаться полезной;
- как быстро становятся доступны новые обучающие данные;
- сколько времени занимает повторное обучение модели;
- сколько стоит развертывание модели;
- каков вклад обновления модели в ценность продукта и достижение целей пользователей.

После обновления рекомендуется выполнить модель на примерах из сквозного и доверительного тестового набора. Важно удостовериться, что выходы такие же, как раньше, или изменились ожидаемым образом. Важно также проверять, что новая модель допускает не намного больше дорогостоящих ошибок, чем предыдущая.

Проверяйте также, что ошибки равномерно распределены между всеми категориями пользователей. Нежелательно, чтобы новая модель работала хуже для пользователей из некоторой миноритарной группы или из конкретного места.

Глава 10

Заключение

В 2020 году машинное обучение стало хорошо отработанным и популярным средством решения деловых задач. То, что раньше было доступно горстке организаций, а остальными считалось «магией», теперь может создать и использовать практически любая организация.

Благодаря открытому исходному коду, краудсорсингу, доступным книгам, онлайн-курсам и выложенным в открытый доступ наборам данных многие ученые, инженеры и просто энтузиасты могут создавать модели машинного обучения. Если повезет, то задачу можно будет решить, написав всего несколько строк кода, как демонстрируется во многих онлайн-пособиях.

Но в проекте машинного обучения многое может пойти наперекосяк. Большинство таких проблем не зависят ни от отработанности технологии, ни от степени понимания аналитиком алгоритма машинного обучения.

Книги, онлайн-пособия и курсы по машинному обучению объясняют, как работают алгоритмы и как применять их к набору данных. Но успех может зависеть и от других факторов. Какие данные вы получаете, можете ли получить их в достаточном количестве, как их подготовить к обучению, какие признаки вы конструируете, является ли решение масштабируемым, пригодным для сопровождения, могут ли злоумышленники манипулировать моделью, допускает ли она много дорогостоящих ошибок – все эти факторы чрезвычайно важны для прикладного проекта машинного обучения.

Но в большинстве книг и курсов по машинному обучению, какими бы объемными они ни были, эти вопросы оставлены для самостоятельного изучения. В некоторых что-то рассматривается, но только в контексте применения к решению конкретной демонстрационной задачи.

Это зияющий пробел в знаниях, и в этой книге я попытался его заполнить.

10.1. Сухой остаток

Что, на мой взгляд, читатель должен вынести, прочитав эту книгу?

Прежде всего ясное понимание того, что все проекты машинного обучения уникальны. Нет универсального рецепта, который работал бы всегда. Как правило, самые важные проблемы должны быть решены еще до написания строки `from sklearn.linear_model import LogisticRegression`: вы должны определить цель, выбрать ориентир, собрать релевантные данные, снабдить их

качественными метками и преобразовать помеченные данные в обучающий, контрольный и тестовый наборы. Оставшиеся проблемы решаются после набора строки `model.fit(X,y)`, для чего применяется анализ ошибок, оценивание модели, проверка того, решает ли она задачу и работает ли лучше, чем имеющееся решение.

Профессиональный аналитик или инженер по машинному обучению понимает, что не все проблемы, деловые и прочие, нужно решать с помощью машинного обучения. На самом деле многие можно решить проще, применив эвристику, поиск в базе данных или разработав традиционное ПО. Пожалуй, не стоит прибегать к машинному обучению, если каждое действие, решение и поведение системы необходимо объяснять. За редкими исключениями, модели машинного обучения представляют собой черные ящики. Они не говорят, ни почему предсказали то, что предсказали, ни почему сегодня не предсказали то, что предсказывали вчера, ни как решить возникшие проблемы.

Далее: если вы не можете найти публично доступный набор данных и решение с открытым исходным кодом, предлагающее именно то, что вам нужно, то машинное обучение – не тот подход, который позволит вывести продукт на рынок в кратчайшие сроки. Иногда данные, необходимые для обучения и сопровождения модели, слишком трудно или даже невозможно получить.

С другой стороны, обучающие данные можно синтезировать с помощью выборки с избытком и приращения данных. Эти методы часто применяются, если данные не сбалансированы.

Прежде чем приступить к сбору данных, задайте себе следующие вопросы: доступны ли данные, велики ли они, пригодны ли для использования, понятны ли и надежны? Хорошие данные содержат достаточно информации для моделирования, покрывают все производственные сценарии, непредвзяты, достаточно велики для обеспечения обобщаемости и не являются результатом самой модели.

А быть может, ваши данные дорого собирать, они сильно смещены, несбалансированы, в них отсутствуют какие-то атрибуты, а метки зашумленные? Прежде чем использовать данные для обучения, необходимо убедиться в их качестве.

Жизненный цикл проекта машинного обучения состоит из следующих этапов: определение цели, сбор и подготовка данных, конструирование признаков, обучение, оценивание, развертывание, выполнение, мониторинг и сопровождение модели. На большинстве этапов может возникнуть просачивание данных. Аналитик должен предвидеть и предотвращать это явление.

Вторым по значимости этапом после подготовки данных является конструирование признаков. Для некоторых типов данных, например текстов на естественном языке, признаки можно генерировать массово, используя такие методы, как мешок слов. Однако наиболее полезные признаки зачастую готовятся вручную аналитиком, сведущим в предметной области. Поставьте себя «на место модели».

Хорошие признаки обладают высокой предсказательной способностью, быстро вычисляются, надежны и некоррелированы. Они цельные, их легко понять и сопровождать. Код выделения признаков – одна из самых важных

частей системы машинного обучения. Он должен быть тщательно и систематически протестирован.

Рекомендуется масштабировать признаки, хранить их в файлах схемы или хранилищах признаков и документировать, а также синхронизировать код, модель и обучающие данные.

Мы можем синтезировать новые признаки путем дискретизации имеющихся, кластеризации обучающих примеров, применения к имеющимся признакам простых преобразований или их комбинирования.

Прежде чем приступить к работе над моделью, убедитесь, что данные согласуются со схемой, а затем разделите их на три набора: обучающий, контрольный и тестовый. Определите достижимый уровень качества и выберите метрику качества. Она должна сводить качество модели к одному числу.

У большинства алгоритмов, моделей и конвейеров машинного обучения имеются гиперпараметры, которые могут оказывать большое влияние на результат обучения. Однако они не обучаются на данных. Аналитик задает значения гиперпараметров в результате настройки. В частности, настройка этих значений контролирует два важных компромисса: между точностью и полнотой и между смещением и дисперсией. Варьируя сложность модели, мы можем достичь так называемой «зоны решений», в которой и смещение, и дисперсия относительно малы. Решение, оптимизирующее метрику качества, обычно ищется в окрестности зоны решений. Поиск на сетке – самый простой и широко распространенный метод настройки гиперпараметров.

Вместо обучения глубокой модели с нуля бывает полезно начать с предобученной модели. Использование предобученной модели для построения собственной называется переносом обучения. Тот факт, что глубокие модели допускают перенос обучения, – одно из их важнейших свойств.

Обучить глубокую модель бывает трудно. Ошибки реализации могут возникать на многих стадиях, от подготовки данных до определения топологии нейронной сети. Рекомендуется начинать с малого. Например, реализуйте простую модель с использованием высокоуровневой библиотеки. Примените значения гиперпараметров по умолчанию к небольшому нормированному набору данных, целиком помещающемуся в оперативную память. Имея первую упрощенную архитектуру модели и набор данных, временно еще уменьшите набор данных до размера мини-пакета. Затем начинайте обучение. Убедитесь, что простая модель способна к переобучению на этом обучающем мини-наборе.

Качество системы машинного обучения можно повысить, воспользовавшись штабелированием моделей. В идеале базовые модели, используемые для штабелирования, должны быть получены с помощью алгоритмов или моделей различной природы, например случайные леса, градиентный бустинг, метод опорных векторов и глубокие модели. Многие реальные производственные системы основаны на штабелированных моделях.

Ошибки машинного обучения могут быть равномерными, т. е. возникающими во всех сценариях с одинаковой частотой, или систематическими, т. е. чаще возникающими в определенных сценариях. Исправив систематическую ошибку, вы исправите проблему сразу во многих примерах.

Качество модели можно улучшить с помощью следующего простого итеративного процесса:

1. Построить модель, используя лучшие найденные на данный момент значения гиперпараметров.
2. Протестировать модель на небольшом подмножестве контрольного набора.
3. Найти самые частые типичные ошибки на этом небольшом контрольном наборе.
4. Сгенерировать новые признаки или добавить обучающие данные, чтобы исправить замеченные ошибки.
5. Повторять, пока не исчезнут часто встречающиеся типичные ошибки.

Модель следует тщательно оценивать перед развертыванием и постоянно после развертывания. Выполняйте офлайн-оценивание, когда модель первоначально обучена на исторических данных. Онлайн-оценивание заключается в тестировании и сравнении моделей в производственной среде с использованием онлайн-данных. Два популярных метода онлайн-оценивания модели: A/B-тестирование и многорукий бандит. Они позволяют определить, какая модель лучше: новая или старая.

Существуют следующие паттерны развертывания моделей: статически (как часть установочного пакета программ), динамически на устройстве пользователя или на сервере и потоком. Дополнительно следует выбрать стратегию развертывания: разовое, немое, канареечное и методом многорукого бандита. У каждого паттерна и стратегии есть свои плюсы и минусы; выбор зависит от особенностей приложения.

Эффективность алгоритма также важно учитывать при развертывании модели. Научные пакеты на Python – NumPy, SciPy, scikit-learn и др. – созданы опытными учеными и инженерами, никогда не упускавшими из вида вопросы эффективности, поэтому многие методы в них написаны на языке C. Избегайте писать собственный код для эксплуатации в производственной среде, если можно воспользоваться популярной и хорошо отработанной библиотекой или пакетом. Для достижения высокой производительности выбирайте подходящие структуры данных и применяйте кеширование.

Для некоторых приложений скорость предсказания критична. В таких случаях производственный код пишется на компилируемом языке, например Java или C/C++. Если аналитик данных построил модель на Python или R, то для развертывания в производственной среде есть несколько вариантов: переписать код на компилируемом языке программирования, использовать стандарт представления моделей, например PMML или PFA, или воспользоваться специализированным движком выполнения типа MLeap.

Модели машинного обучения выполняются в пакетном режиме или по запросу. В случае выполнения по запросу модель обычно обернута REST API. Для обслуживания запросов к модели часто применяется потоковая архитектура.

Архитектура программной системы, предназначенной для работы в реальных условиях, должна учитывать возможность ошибок, изменения и человеческую природу. Модель следует постоянно мониторить. Мониторинг

должен проверять, что модель правильно обслуживает запросы, а ее качество остается в допустимых пределах.

Важно протоколировать достаточно информации, чтобы можно было воспроизвести любое странное поведение в процессе будущего анализа. Если модель обслуживает пользователя какого-то графического интерфейса, то важно сохранять контекст пользователя в момент обращения к модели.

Некоторые пользователи могут использовать вашу модель для достижения собственных целей. Чтобы предотвратить неправомерное использование, не доверяйте данным от некоторого пользователя, если похожие данные не поступают от многих пользователей. Назначайте каждому пользователю репутационную оценку и не доверяйте данным, полученным от пользователей с низкой репутацией. Классифицируйте поведение пользователя как нормальное или аномальное и постепенно увеличивайте паузы перед отправкой ответа на запрос, а при необходимости блокируйте некоторых пользователей.

Регулярно обновляйте модель, анализируя поведение пользователей и входные данные, чтобы сделать модель более робастной. После обновления примените новую модель к сквозному и доверительному тестовому набору. Убедитесь, что выходы такие же, как раньше, или изменились ожидаемым образом. Проверьте, что новая модель допускает не намного больше дорогостоящих ошибок, чем предыдущая. Убедитесь, что ошибки равномерно распределены между всеми категориями пользователей. Нежелательно, чтобы новая модель работала хуже для пользователей из некоторой миноритарной группы или из конкретного места.

Книга закончилась, но ваше обучение на этом не кончается. Инженерия машинного обучения – сравнительно новый раздел программной инженерии. Я уверен, что в ближайшие годы появятся новые передовые практики, библиотеки и каркасы, упрощающие или подкрепляющие этапы подготовки данных, оценивания, развертывания, выполнения и мониторинга моделей, – наличие онлайн-публикаций и открытого исходного кода тому порукой. Подпишитесь на мой список рассылки на сопроводительном сайте этой книги <http://www.mlebook.com>. Тогда вы будете регулярно получать релевантные ссылки.

10.2. Что еще почитать

Есть много прекрасных книг по машинному обучению и искусственному интеллекту. Ниже упомянуто лишь несколько из них.

Если вас интересует практическое применение машинного обучения на Python, то могу рекомендовать две книги:

- Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd edition). O'Reilly Media, 2019¹;

¹ Орельен Жерон. Прикладное машинное обучение с помощью Scikit-Learn, Keras и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем. 2-е изд. Диалектика; Вильямс, 2020.

- *Sebastian Raschka*. Python Machine Learning (3rd edition). Packt Publishing, 2019¹.

Что касается R, то лучшей, на мой взгляд, является книга: *Brett Lantz*. Machine Learning with R. Packt Publishing, 2019.

Читателям, желающим глубже понять математику, стоящую за различными алгоритмами машинного обучения, рекомендую следующие книги:

- *Christopher Bishop*. Pattern Recognition and Machine Learning. Springer, 2006²;
- *Gareth James et al.* An Introduction to Statistical Learning. Springer, 2013. В качестве углубленного курса машинного обучения рекомендую:
- *Michael Nielsen*. Neural Networks and Deep Learning. Онлайн, 2005;
- *David Foster*. Generative Deep Learning. O'Reilly Media, 2019³.

Если ваши амбиции простираются далеко за пределы машинного обучения и вы хотите освоить весь пласт знаний, связанных с искусственным интеллектом, то нет ничего лучше книги: *Stuart Russell and Peter Norvig*. Artificial Intelligence: A Modern Approach (4th Edition). Pearson, 2020), или AIMA⁴.

10.3. Благодарности

Добиться высокого качества книги было бы невозможно без редакторов-добровольцев. Особенно хочу отметить следующих читателей за их систематическую помощь: Alexander Sack, Ana Fotina, Francesco Rinarelli, Yonas Mitike Kassa, Kelvin Sundli, Idris Aleem, Tim Flocke.

Благодарю научных консультантов, Веронику Тремблей и Максимилиана Худльбергера, за рецензирование и корректуру главы, посвященной оцениванию моделей. Я также признателен Кэсси Козырьков за внимательное и критическое прочтение раздела о статистических критериях.

Перечислю других замечательных людей, которым я благодарен за помощь: Jean Santos, Carlos Azevedo, Zakarie Hashi, Tridib Dutta, Zakariya Abu-Grin, Suhel Khan, Brad Ezard, Cole Holcomb, Oliver Proud, Michael Schock, Fernando Hannaka, Ayla Khan, Varuna Esver, Stephen Fox, Brad Klassen, Felipe Duque, Alexandre Mundim, John Hill, Ryan Volpi, Gaurish Katlana, Harsha Srivatsa, Agrita Garnizone, Shyambhu Mukherjee, Christopher Thompson, Sylvain Truong, Niklas Hansson, Zhihao Wu, Max Schumacher, Piers Casimir, Harry Ritchie, Marko Peltojoki, Gregory V., Win Pet, Yihwa Kim, Timothée Bernard, Marwen Sallem, Daniel Bourguet, Aliza Rubenstein, Alice O., Juan Carlo Rebanal, Haider Al-Tahan, Josh Cooper, Venkata Yerubandi, Mahendren S., Abhijit Kumar, Mathieu Bouchard,

¹ *Себастьян Рашка*. Python и машинное обучение. 3-е изд. Диалектика; Вильямс, 2020.

² *Кристофер М. Бишоп*. Распознавание образов и машинное обучение. Диалектика; Вильямс, 2020.

³ *Дэвид Фостер*. Генеративное глубокое обучение. Питер, 2020.

⁴ *Стюарт Рассел, Питер Норvig*. Искусственный интеллект: современный подход. Диалектика; Вильямс, 2020, 2021.

Yacin Bahi, Samir Char, Luis Leopoldo Perez, Mitchell DeHaven, Martin Gubri, Guillermo Santamaría, Mustafa Murat Arat, Rex Donahey, Nathaniel Netirungroj, Aliza Rubenstein , Rahima Karimova, Darwin Brochero, Vaheid Wallets, Bharat Raghunathan, Carlos Salas, Ji Hui Yang, Jonas Atarust, Siddarth Sampangi, Utkarsh Mittal, Felipe Antunes, Larysa Visengeriyeva, Sorin Gatea, Mattia Pancerasa, Victor Zabalza, Dibyendu Mandal, James Hoover.

Предметный указатель

A

A/B-тестирование, 225, 227, 285
Adam, 192
ADASYN, 86
Akka Streams, 252
AlexNet, 131
Amazon S3, 93
Apache Flink, 251
Apache Kafka, 272
Apache Kafka Streams, 252
Apache Samza, 252
Apache Spark, 251, 260
Apache Storm, 251
ARFF, 91
AUC, 165

B

BERT, 85, 183
BLOB-объект, 93
Boruta, 120

C

CephFS, 92
CIFAR-10, 84
CityHash, 109
Core ML, 246
cProfile, 259
cuML, 284

D

doc2vec, 85, 130
Docker, 249

DVC, 95

E

ELMo, 183

F

F-мера, 162, 184
Fargate, 249
FastAPI, 248
fastText, 130
Flask, 248

G

G-критерий, 228
GAN, 84
GCS, 93, 95
Git, 95
git-сигнатура, 95
Git Large File Storage, 95
Google Cloud Storage, 93
Google Kubernetes Engine, 249
GoogLeNet, 183
gRPC, 247
GRU, 183

H

Hadrian, 263
HDFS, 92

I

ImageNet-2012, 84
InceptionV3, 183

iterators, 259

J

Jenkins, 109

joblib, 262

JPMML, 263

K

к ближайших соседей, 30, 87, 151, 155

Kaggle, 120

Keras, 213, 246

kNN, 30

Kubernetes, 249

L

LDA, 109

LFS, 95

LIBLINEAR, 91

LIBSVM, 91

lineprof, 259

LSA, 109

LSTM, 183

двунаправленная, 184

M

MAB, 255

MD5, 109

MdAE, 159

Memcached, 261

Mercurial, 95

MLeap, 263

Momentum, 192

multiprocessing, 260

MurmurHash3, 109

N

n-грамма, 104

NFS, 92

Numba, 218, 260

NumPy, 218, 262

O

О большое нотация, 257

P

Pachyderm, 95

Packrat, 221

parallel, 260

PCA, 133, 158

инкрементный, 133

PFA, 263

Pickle, 92, 174, 261

PII, 54

Plumber, 248

PMML, 262

pqR, 218

PULSE, 63

PyPy, 218, 260

R

RabbitMQ, 272

RBF, 178

Redis, 261

ReLU, 186, 197

ResNet50, 183

REST API, 247, 272

RMSProp, 192

ROC-кривая, 164

S

scikit-learn, 123, 158, 246, 262

SciPy, 218

SkLearn2PMML, 263

SMOTE, 86

softmax, 130, 186, 197

SPE, 251

SPL, 252

SUT, 240

SVD, 109

SVM, 30, 70, 155, 158, 200, 203

T

TCP/IP, 93

TensorFlow, 246

TensorFlow.js, 246

TF-IDF, 104, 109, 151

thundersvm, 284

TPR, 242

U

UCB1, 234
UMAP, 133, 210

V

VGG, 183
VGG16, 183
VGG19, 183
virtualenv, 221

W

word2vec, 65, 128

X

XGBoost, 246

Z

z-оценка, 121, 135

A

Автокодировщик, 71, 133, 210
Автомасштабирование, 272
Автомасштабируемая группа, 247
Аккуратные данные, 26
Актуальность модели, 284
Алгоритм
 кластеризации на основе
 центроидов, 87
 правила нуля, 150
 случайного предсказания, 150
Алгоритм обучения
 без учителя, 23
 инкрементного, 155
 классификатора, 29
 на основе модели, 30
 на основе экземпляров, 30
 регрессии, 30
 с учителем, 23
Альтернативная гипотеза, 228
Анализ
 алгоритмов, 257
 оттока клиентов, 125
 ошибок, 208

по частям, 212

Ансамблевое обучение, 199, 208
Ансамбль туркеров, 151
Атрибут, 26
Ахо–Корасик алгоритм, 101

Б

База данных, 93
 реляционная, 125
Базовая модель, 199
Балансировщик нагрузки, 247
Бессерверные вычисления, 250
Библиотека потоковой обработки, 252
Биграмма, 104
Блок
 линейной ректификации, 186, 197
 управляемый рекуррентный, 183
Брокер сообщений, 272, 286
Бутстреп, 237
Бэггинг, 121

В

Вектор, 19
Вектор признаков, 22
Верность, 42, 150, 162
 средняя, 163, 205
 с учетом стоимости, 163
Винсоризация, 134
Виртуальная машина, 221
Воспроизводимость, 97, 139, 148, 221
Временной ряд, 112
 классический, 112
Выборка
 вероятностная, 88
 интервальная, 90
 невероятностная, 89
 простая случайная, 89
 с возвращением, 237
 с избытком, 86
 систематическая, 56, 90
 с недостатком, 42, 87
 кластерная, 87
 стратифицированная, 56, 90, 282
Выборочная дисперсия, 126
Выборочная проверка, 156

Выборочное среднее, 105, 126, 135
Выделение признаков, 268

Г

Генератор, 259
Гипероним, 84
Гиперпараметр, 28, 29, 179
Главная компонента, 133
Голосование, 200
Градиент, 189
Градиентный бустинг, 199
Градиентный спуск, 187
 стохастический, 188
 стохастический на
 мини-пакетах, 191
Группировка, 124
Групповое разбиение, 80

Д

Данные
 аннотирование, 40
 временных рядов, 79, 112
 зарезервированные, 159
 косвенное использование, 25, 101
 неполные, 57
 нерепрезентативные, 57
 несбалансированные, 57, 85, 163, 203
 о взаимодействии, 75
 первичные, 26
 прямое использование, 25, 101
 разреженные, 92
 с истекшим сроком действия, 57
 структурированные, 93
Движок потоковой обработки, 251, 267
Двойное применение, 279
Двойное снижение в глубокой сети, 177
Двусторонний критерий равенства, 122
Диапазон допустимых значений, 255, 268
Дилемма исследования и использования, 234

Дискретизация, 127
Дисперсия высокая, 177
Дифференцирование, 189
Длинный хвост, 120
Документ о жизненном цикле данных, 96
Долгая краткосрочная память, 114, 183
Дубликаты, 57

Е

Евклидова норма, 21
Евклидово расстояние, 21, 70, 85, 87, 241

З

Задача восприятия, 32
Запоминание, 260
Зашумленная предварительная разметка, 61
Защищенные атрибуты, 242

И

Измерение, 20
Интервал, 188
 открытый, 188
 предсказания, 239
 статистический, 236
Истинно отрицательный результат, 160
Истинно положительный результат, 160

К

Калибровка
 модели, 280
 признаков, 105
Каппа Коэна, 163, 184, 195, 205, 213
Каскадирование
 коррекций, 217
 моделей, 217, 288
Катастрофическое забывание, 216
Кеширование, 260
Класс, 23, 29

Классификация, 22, 29
 бинарная, 29, 185
 биномиальная, 29
 многозначная, 154, 185
 многоклассовая, 29, 153, 162, 185, 197
 мультиномиальная, 29
 Кластеризация, 23, 90, 126
 к средних, 126
 Кодирование
 средним, 105, 136
 унитарное, 102, 125, 128, 153, 185, 197
 Коллизия, 109
 Колмогорова–Смирнова
 критерий, 280
 Компромисс
 между смещением
 и дисперсией, 177
 между точностью и полнотой, 161
 Конвейер, 28, 158, 271
 Конструирование признаков, 26
 Контейнер, 221
 Контекст, 270, 282
 Контрольная точка, 193
 Корректность модели, 240
 Корреляция, 31, 118, 242
 Кривая
 калибровки, 206
 обучения, 55
 Критерий независимости
 хи-квадрат, 280

Л

Латентное распределение
 Дирихле, 109
 Латентный семантический
 анализ, 109
 Лексемизатор, 102
 Лексемизация, 102
 Линейная регрессия, 155
 Линейное ядро, 150
 Логарифм отношения шансов, 105
 Логистическая сигмоида, 196
 Ложноотрицательные
 результаты, 160

Ложноположительные
 результаты, 160
 частота, 164

М

Магическое число, 57
 Масштабирование
 до нуля, 249
 признаков, 133
 Матрица, 20
 неточностей, 160, 242
 Машинное обучение, 21
 Медиана, 114
 Медианное абсолютное
 отклонение, 114, 159
 Межквартильный размах, 114
 Метка, 22, 29
 косвенная, 59
 отсроченная, 59
 Метод
 главных компонент, 133, 158
 локтя, 127
 опорных векторов, 30, 70, 150, 158, 203, 207
 Метрика качества, 43, 45, 149, 159, 255, 268
 важная для бизнеса, 216
 Механический турок, 151
 Мешок
 слов, 102, 109, 119, 136, 150, 154, 185
 n-грамм, 104
 Минимум
 глобальный, 188
 локальный, 188
 Мини-пакет, 191, 214
 Многорукий бандит, 233, 255, 285
 Многослойный перцептрон, 127, 129
 Множество, 20
 Моделирование методом
 Монте-Карло, 230
 Модель, 29
 ансамблевая, 155
 байесовская, 109
 предобученная, 183
 среднего, 159

статистическая, 22
 Мутационное тестирование, 240
 Мягкий зазор, 204

Н

Набор
 доверительный сквозной, 268
 доверительный тестовый, 255, 268, 280
 зарезервированный, 28, 78
 контрольный, 27, 78, 152, 156
 обучающий, 27, 78, 152
 сквозной, 255, 280
 тестовый, 27, 78, 152
 Наивный байесовский классификатор, 155
 Накопительный выигрыш, 166
 дисконтированный, 166
 нормированный
 дисконтированный, 167
 Настройка гиперпараметров, 113, 169, 208
 Недообучение, 159, 175
 Нейронная сеть, 30, 70, 127, 208
 глубокая, 30, 155
 остаточная, 183
 полносвязная, 129
 рекуррентная, 130
 сверточная, 114, 130, 151, 183
 управляемая рекуррентная, 183
 Нейронное покрытие, 239
 Необъективность
 экспериментатора, 66, 68
 Несбалансированность классов, 85
 Нормировка, 134
 пакетная, 180, 193
 по среднему значению, 135
 по z-оценке, 135
 Нулевая гипотеза, 122, 228

О

Область
 значений, 188
 определения, 188
 Обнаружение выбросов, 24

Обобщаемость, 123, 159
 Обратное распространение, 190
 Обратный перевод, 85
 Обучение
 ансамблевое, 199
 без учителя, 23
 глубокое, 30, 63
 модели, 30
 поверхностное, 30, 113, 130, 195
 решающих деревьев, 155
 с подкреплением, 24
 с учителем, 22
 Объединение, 21
 Объект, 93
 Объяснимость, 180
 Один против остальных, 30, 206
 Ожидаемая стоимость ошибки, 272, 274
 Озеро данных, 94
 Оптимизация при разумной достаточности, 162
 Ориентир, 28, 149, 159
 Отбор признаков, 180
 Отиаи коэффициент, 85, 87, 128
 Отношение шансов, 105
 Отрицательное логарифмическое правдоподобие, 130, 197
 Отсечение, 134
 Отсутствующие значения, 57
 Оценивание, 31, 158
 Оценивание модели
 онлайновое, 226
 офлайновое, 225

П

Пакетная нормировка, 208
 Параметр, 29, 30
 Паттерн развертывания, 244, 266
 гибридного, 251
 потокowego, 251
 Перекрестная проверка, 27, 79, 172
 пятигрупповая, 172
 Перекрестная энтропия, 197
 бинарная, 185, 197
 категориальная, 185

Перенос обучения, 184, 195, 197, 208, 217
 Переобучение, 63, 159, 176, 214
 Пересечение, 21
 Петля обратной связи, 59, 73
 План уменьшения скорости обучения, 191
 временной, 191
 шаговый, 192
 экспоненциальный, 192
 Платта масштабирование, 207
 Площадь под ROC-кривой, 165
 Погружение, 71, 114, 196
 документа, 84
 слова, 65, 84, 128
 Подсеть, 196
 Подстановка данных, 57, 63
 Подсчет интервалов, 105
 Поиск
 на сетке, 70, 169
 с измельчением, 170
 Политика, 24
 Полнота, 161, 164, 274
 Понижение размерности, 24
 Порождающая состязательная сеть, 84
 Потокковое развертывание модели, 251, 286
 Поток событий, 112
 Правило дифференцирования сложной функции, 189
 Предсказание оттока клиентов, 59, 284
 Предсказательная сила, 127
 Предсказательная способность, 116, 117, 176, 208
 низкая, 69
 Признак, 22
 циклический, 119
 Пример
 непомеченный, 23, 29
 первичный, 27
 помеченный, 22, 29
 состязательный, 84
 Приращение данных, 82, 97, 180, 193, 208

Проверка статистических гипотез, 228
 Производная, 132, 189
 частная, 189
 Прореживание, 180, 192, 194
 Просачивание
 групповое, 79
 данных, 28, 58, 71, 75, 201, 219
 целей, 71
 цели, 58

Р

Радиально-базисная функция, 177
 Развертывание
 канареечное, 250, 254
 немое, 254
 разовое, 253
 Размерность, 22
 Разметчик, 46, 66, 279, 286
 Разреженность, 104, 117, 123, 180
 Ранняя остановка, 180, 193
 Распределение
 нормальное, 187
 равномерное, 187
 стандартное нормальное, 135
 Распределение по интервалам, 124
 Регрессия, 22, 30
 изотоническая, 207
 Регуляризация, 123, 179, 208
 гребневая, 180
 эластичная сеть, 180
 L1, 179, 192
 L2, 179, 192
 lasso, 180
 Рекомендательная система, 281
 Решающая граница, 71
 Решающее дерево, 203
 РНС, 130, 196
 Робастность модели, 241

С

Связующий код, 219
 Сдвиг
 априорного распределения, 202
 ковариат, 202

- распределения, 65, 70, 143, 152, 202, 208, 219, 227, 280, 286
 - Сериализация данных, 92
 - Сильная модель, 199
 - Сингулярное разложение, 109
 - Синоним, 84
 - Синусно-косинусное преобразование, 106
 - Систематическая ошибка, 66, 68, 209, 279
 - Скаляр, 19
 - Скалярное произведение, 218
 - Скипграмма, 128
 - СКО, 159
 - Скорость
 - обучения, 190
 - уменьшения, 191
 - Скрещивание признаков, 127
 - Скрытая петля обратной связи, 141, 269, 279
 - Слабая модель, 199
 - Словарь, 259
 - Сложность модели, 150
 - Слой
 - погружения, 128
 - полносвязный, 131, 183
 - сверточный, 183
 - тах-пулинга, 183
 - Случайное начальное значение, 221
 - Случайный лес, 120, 199, 203, 237
 - Случайный поиск, 170
 - Смена концепта, 70, 202, 276, 279
 - Смесь, 109
 - Смешивание, 83
 - Смещение
 - выборки, 65, 68
 - высокое, 175
 - данных, 63
 - из-за отсутствующей переменной, 65, 67
 - из-за предубеждения, 65, 68
 - из-за спонсорства, 65, 68
 - из-за стереотипов, 65, 68
 - из-за финансирования, 65
 - низкое, 175
 - отбора, 63, 67
 - пометки, 66, 68
 - предсказания, 280
 - самоотбора, 64, 67
 - СНС, 114, 130, 131, 151, 183, 196
 - Состязательная проверка, 202
 - Справедливость, 69
 - модели, 242
 - Среднее арифметическое, 114
 - Среднеквадратическая ошибка, 159, 185
 - Стандартизация, 135, 193
 - Стандартное отклонение, 114, 126, 135
 - Статистический паритет, 242
 - Статистический тест, 120
 - Стоп-слово, 123
 - Стратегия инициализации параметров, 187
 - все единицы, 187
 - все нули, 187
 - нормальная Ксавье, 187
 - равномерная Ксавье, 187
 - случайная нормальная, 187, 195, 214
 - случайная равномерная, 187, 195
 - Строка, 93
 - Структура данных, 259
 - СУБД, 93
 - Схема
 - при записи, 94
 - при чтении, 94
 - Сходимость, 190
- Т**
- Таблица, 93
 - сопряженности, 106
 - Тензор, 27
 - Теплый старт, 216
 - Тестируемая программа, 240
 - Томека связь, 87
 - Топологический граф, 253
 - Топология
 - обработки данных, 252
 - сети, 183
 - Точечный процесс, 112
 - Точность, 161, 274

Трансформер, 114, 177, 183

У

Уменьшение весов, 180

Унитарное кодирование, 128

Уровень

достоверности, 237

хранения, 92

Усреднение, 200

Усталость пользователя, 278

Ф

Файл, 92

схемы, 136, 148

Файловая система, 92

локальная, 92

распределенная, 92

Фактор отторжения, 278

Функция, 188

активации, 130

стоимости, 45, 86, 130, 159, 185, 222

хеширования, 108

Х

Хеширование признаков, 108, 142

Хеш-таблица, 259

Хи-квадрат распределение, 229

Хорошо откалиброванная модель, 205

Хранилище

объектов, 93

признаков, 138

Ц

Целевой показатель, 22

Цель машинного обучения, 36, 43

Ч

Частота

истинно положительных

результатов, 164, 242

паттерна ошибок, 210

почти правильных

предсказаний, 160

Численная устойчивость, 280

Численное переполнение, 134

Чувствительные атрибуты, 242

Ш

Штабелирование моделей, 77, 200

Шум, 62

гауссов, 83

Э

Эйлера число, 186, 192

Эксперимент, 42

Экстрактор признаков, 198, 271

Эпоха, 190

Эффективность алгоритма, 257

Я

Ядерная функция, 70

Книги издательства «ДМК ПРЕСС»
можно купить оптом и в розницу
в книготорговой компании «Галактика»
(представляет интересы издательств
«ДМК ПРЕСС», «СОЛОН ПРЕСС», «КТК Галактика»).

Адрес: г. Москва, пр. Андропова, 38, оф. 10;
тел.: **(499) 782-38-89**, электронная почта: **books@aliants-kniga.ru**.

При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.

Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: <http://www.galaktika-dmk.com/>.

Андрей Бурков

Инженерия машинного обучения

Главный редактор	<i>Мовчан Д. А.</i> dmkpress@gmail.com
Зам. главного редактора	<i>Сенченкова Е. А.</i>
Перевод	<i>Снастин А. В.</i>
Корректор	<i>Синяева Г. И.</i>
Верстка	<i>Чаннова А. А.</i>
Дизайн обложки	<i>Мовчан А. Г.</i>

Гарнитура PT Serif. Печать цифровая.
Усл. печ. л. 24,86. Тираж 200 экз.

Веб-сайт издательства: www.dmkpress.com

«Если вы собираетесь использовать машинное обучение для решения крупномасштабных бизнес-задач, могу только порадоваться, что вы наткнулись на эту книгу. Читайте с удовольствием!»

*Кэсси Козырьков,
главный специалист по теории принятия решений в Google*

«Фундаментальная работа о практическом построении моделей машинного обучения (МО) и их развертывании в производственной среде. Подоспела как раз в тот момент, когда компании начали прозревать и осознавать, что для того, чтобы решение, основанное на машинном обучении, заработало, необходимы осознанные усилия инженеров и знакомство с передовыми практиками. Еще одна замечательная книга от Андрея!»

*Каролис Урбонас,
начальник отдела машинного обучения в Amazon*

Новая книга Андрея Буркова, руководителя группы МО в компании Gartner и автора «Машинного обучения без лишних слов», мирового бестселлера, изданного на 11-ти языках, – одно из наиболее полных на сегодня изданий по прикладному искусственному интеллекту (ИИ). Содержит множество рекомендаций и паттернов проектирования надежных и масштабируемых решений в области машинного обучения.

Книга основана на собственном 15-летнем опыте автора в решении промышленных задач с помощью ИИ, а также на опубликованных работах лидеров индустрии.

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
e-mail: books@aliens-kniga.ru

ДМК
пресс
издательство
www.dmk.pf

ISBN 978-5-93700-125-2



9 785937 001252 >