Содержание

Предисловие

В 2012 году я опубликовал 1200-страничную книгу под названием «Machine Learning: A Probabilistic Perspective», в которой сложившаяся на тот момент дисциплина машинного обучения (МО) достаточно полно рассматривалась сквозь объединяющую призму вероятностного моделирования. Книга была хорошо принята и получила премию де Грута (https://bayesian.org/project/degroot-prize/) в 2013 году.

2012-й также принято считать началом «революции глубокого обучения». Термином «глубокое обучение» называют раздел МО, основанный на нейронных сетях с большим количеством слоев (отсюда и слово «глубокий»). Хотя базовая технология к тому времени существовала уже много лет, именно в 2012 году в работе [KSH12] глубокие нейронные сети (ГНС) выиграли конкурс по классификации изображений ImageNet с таким отрывом, что это привлекло внимание профессионального сообщества. Примерно в то же время был достигнут прогресс в таких трудных задачах, как распознавание речи (см., например, [Cir+10; Cir+11; Hin+12]). Эти прорывы стали возможными благодаря достижениям в развитии оборудования (в частности, перепрофилированию быстрых графических процессоров, GPU, с видеоигр на МО), технологиях сбора данных (в частности, применению краудсорсинговых инструментов типа «Механического турка» от Amazon для сбора больших размеченных наборов данных, как в ImageNet), а также различным новым алгоритмическим идеям; некоторые из них мы рассмотрим в этой книге.

Начиная с 2012 года, область глубокого обучения развивалась лавинообразно, новые результаты появлялись со все возрастающей скоростью. Столь же лавинообразно рос и интерес к этой области, подогреваемый коммерческим успехом технологии. Поэтому в 2018 году я решил написать второе издание своей книги и попытаться подвести какой-то итог.

К марту 2020 года черновой вариант второго издания разросся до 1600 страниц, а еще много тем оставались неосвещенными. В результате издательство MIT Press порекомендовало мне разбить книгу на два тома. Но тут разразилась пандемия COVID-19. Я решил на время отвлечься от написания книги и поучаствовать в разработке алгоритма оценки рисков для разрабатываемого Google приложения, уведомляющего о подверженности заражению [MKS21], а также в других проектах, связанных с прогнозированием [Wah+21]. Однако к осени 2020 года я решил вернуться к работе над книгой.

Чтобы наверстать упущенное, я попросил нескольких коллег помочь мне в написании различных разделов (см. благодарности ниже). В результате появились две новые книги «Вероятностное машинное обучение: введение», которую вы сейчас читаете, и «Вероятностное машинное обучение: дополнительные вопросы», которая является ее продолжением [Mur22]. Надеюсь, что в совокупности они дают довольно полное представление о состоянии дел в МО в 2021 году – сквозь ту же объединяющую призму вероятностного

моделирования и байесовской теории принятия решений, которая использовалась в книге 2012 года.

Почти весь материал первого издания сохранен, но теперь он более-менее равномерно распределен между двумя новыми книгами. Кроме того, в каждую книгу включено много дополнительных тем из области глубокого обучения и других разделов теории, например порождающие модели, вариационный вывод и обучение с подкреплением.

Чтобы сделать эту вводную книгу более замкнутой и полезной студентам, я добавил ряд сведений по базовым дисциплинам, в частности оптимизации и линейной алгебре, которых за скудостью места не было в издании 2012 года.

Материал повышенной сложности, который можно опустить в курсе вводного уровня, помечен звездочкой * в названии раздела или главы.

В конце некоторых глав имеются упражнения. Решения упражнений можно найти в сети. Дополнительные учебные материалы (например, рисунки и слайды) см. на сайте книги probml.ai (https://probml.github.io/pml-book/).

Еще одно существенное изменение – использование Python вместо Matlab во всех примерах программ. (В будущем мы, возможно, создадим версию кода на Julia.) В новом коде используются такие стандартные Python-библиотеки, как NumPy (https://numpy.org/), Scikit-learn (https://scikit-learn.org/stable/), JAX (https://github.com/google/jax), PyTorch (https://pytorch.org/), TensorFlow (https://www.tensorflow.org/), PyMC3 (https://docs.pymc.io/en/v3/) и др. О том, как использовать код, см. на странице по адресу https://probml.github.io/pml-book/.

Благодарности

Я выражаю благодарность следующим людям, помогавшим мне при написании книги:

- Зико Колтеру (Zico Kolter), помогавшему писать части главы 7 («Линейная алгебра»);
- Фредерику Кунстнеру (Frederik Kunstner), Си И Менгу (Si Yi Meng), Аарону Мишкину (Aaron Mishkin), Шарану Васвани (Sharan Vaswani) и Марку Шмидту (Mark Schmidt), помогавшим писать части главы 8 («Оптимизация»);
- Матью Блонделю (Mathieu Blondel), помогавшему писать раздел 13.3 («Обратное распространение»);
- Кшиштофу Хороманski (Krzysztof Choromanski), помогавшему писать раздел 15.6 («Эффективные преобразователи»*);
- Колину Раффелю (Colin Raffel), помогавшему писать раздел 19.2 («Перенос обучения») и раздел 19.3 («Обучение с частичным привлечением учителя»);
- Брайану Перощи (Bryan Perozzi), Сами Абу Эль Хайджа (Sami Abu-El-Haija) и Инес Чами (Ines Chami), помогавшим писать главу 23 («Погружения графов»*);
- Джону Фэнсу (John Fearns) и Питеру Черно (Peter Cerno) за внимательную вычитку книги;

- многочисленных членов сообщества github за поиск опечаток и других ошибок (см. перечень таковых по адресу https://github.com/probml/pml-book/issues?q=is:issue);
- четырем анонимным рецензентам, выбранным MIT Press;
- Махмуду Солиману (Mahmoud Soliman), который написал весь код, связующий latex, colab, github и пр., и поведал мне о GCP и TPU;
- всей когорте студентов, работавших над кодом книги на Google Summer of Code 2021 года; это Алейна Кара (Aleyna Kara), Срикар Джилугу (Srikar Jilugu), Дришти Патель (Drishti Patel), Минь Лиан Анг (Ming Liang Ang), Жерардо Дюран-Мартен (Gerardo Durán-Martín) (см. перечень их трудов по адресу https://probml.github.io/pml-book/gsoc2021.html);
- многочисленным членам сообщества github за предложенный ими код (см. https://github.com/probml/pyprobml#acknowledgements);
- авторам работ [Zha+20], [Gér17] и [Mar18], разрешившим мне использовать или модифицировать части открытого исходного кода, включенного в их замечательные книги;
- моему начальнику в Google, Дугу Эку (Doug Eck), который позволил мне тратить принадлежащее компании время на написание этой книги;
- своей жене Маргарет, позволившей мне тратить принадлежащее семье время на эту книгу.

Об обложке

На обложке изображена нейронная сеть (глава 13), которая используется для отнесения рукописной цифры x к одному из десяти классов меток $y \in \{0, 1, \ldots, 9\}$. Гистограмма справа – вывод модели, соответствующий условному распределению вероятности $p(y|x)$.

Кэвин Патрик Мэрфи
Паль-Альто, Калифорния
август 2021

Глава 1

Введение

1.1. Что такое машинное обучение?

Популярное определение **машинного обучения**, или **МО**, принадлежащее Тому Митчеллу [Mit97], звучит следующим образом:

> Говорят, что компьютерная программа обучается на опыте E относительно некоторого класса задач T и меры качества P, если ее качество на задачах, принадлежащих T, измеренное в соответствии с P, улучшается с увеличением опыта E.

Таким образом, существует много видов машинного обучения, зависящих от природы задачи T, решению которой мы хотим обучить систему, природы меры качества P, используемой для оценки работы системы, и природы обучающего сигнала, или опыта E, на котором обучается система.

В этой книге мы будем рассматривать наиболее распространенные типы МО, но с **вероятностной точки зрения**. Грубо говоря, это означает, что все неизвестные переменные (например, предсказания будущего значения некоторой величины, скажем температуры на завтра или параметров некоторой модели) рассматриваются как **случайные величины**, имеющие **распределение вероятностей**, которое описывает взвешенное множество возможных значений переменной (см. краткое введение в основы теории вероятностей в главе 2).

Есть две основные причины, чтобы отдать предпочтение вероятностному подходу. Во-первых, он оптимален для **принятия решений в условиях неопределенности**, это будет объяснено в разделе 5.1. Во-вторых, вероятностное моделирование – язык, используемый в большинстве других областей науки и техники, так что мы получаем единую систему понятий. Шакир Мохамед (научный сотрудник компании DeepMind)[1] говорил:

> Почти все машинное обучение можно представить в вероятностных терминах, так что вероятностный подход является фундаментальным. Такое представление, конечно, не единственное. Но именно оно позволяет

[1] Источник: слайд 2 на странице https://bit.ly/3pyHyPn.

связать машинное обучение с другими областями вычислительных наук, будь то стохастическая оптимизация, теория управления, исследование операций, эконометрика, теория информации, статистическая физика или биостатистика. Уже по этой причине умение рассуждать в вероятностных терминах обязательно.

1.2. Обучение с учителем

Самая распространенная форма МО – **обучение с учителем**. В этом случае задача T заключается в том, чтобы обучиться отображению f множества входов $x \in \mathcal{X}$ в множество выходов $y \in \mathcal{Y}$. Входы x называются также **признаками**, **ковариатами** или **предикторами**; часто это числовой вектор фиксированной размерности, например рост и вес человека или пиксели изображения. В таком случае $\mathcal{X} = \mathbb{R}^D$, где D – размерность вектора (т. е. количество входных признаков). Выход y называется также **меткой**, **целью** или **откликом**[1]. Опыт E задается в виде множества N пар вход–выход $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^N$, которое называет **обучающим набором** (а N – **размером выборки**). Мера качества P зависит от типа предсказываемого выхода, мы обсудим это ниже.

1.2.1. Классификация

В задачах **классификации** пространство выходов C неупорядочено, а метки называются **классами**, $\mathcal{Y} = \{1, 2, \ldots, C\}$. Проблема предсказания метки класса заданного входа называется также **распознаванием образов**. (Если имеется всего два класса, часто обозначаемых $y \in \{0, 1\}$ или $y \in \{-1, +1\}$, то говорят о **бинарной классификации**.)

1.2.1.1. Пример: классификация ирисов

В качестве примера рассмотрим проблему классификации ирисов с отнесением к трем видам: щетинистый (Setosa), разноцветный (Versicolor) и виргинский (Virginica). На рис. 1.1 показано по одному примеру из каждого класса.

В задаче **классификации изображений** пространством входов $\mathcal{X}$ является множество изображений, имеющее очень высокую размерность: для цветного изображения с $C = 3$ каналами (например, в формате RGB) и $D_1 \times D_2$ пикселями имеем $\mathcal{X} = \mathbb{R}^D$, где $D = C \times D_1 \times D_2$. (На практике яркость пикселя представлена целым числом, обычно в диапазоне $\{0, 1, \ldots, 255\}$, но для простоты обозначений мы предполагаем, что входы – вещественные числа.) Обучиться отображению $f : \mathcal{X} \to \mathcal{Y}$ изображений в метки очень трудно, как легко видеть из рис. 1.2. Однако эту задачу все же можно решить с помощью

[1] Иногда (например, в Python пакете statsmodels [https://www.statsmodels.org/devel/endog_exog.html]) входы x называются **экзогенными переменными**, а выходы y – **эндогенными переменными**.

функций специального вида, например **сверточной нейронной сети** (СНС) (convolutional neural network – CNN), которую мы будем обсуждать в разделе 14.1.

Рис. 1.1 ❖ Три типа ирисов: щетинистый (Setosa), разноцветный (Versicolor) и виргинский (Virginica). Печатается с разрешения Денниса Крэмба и компании SIGNA

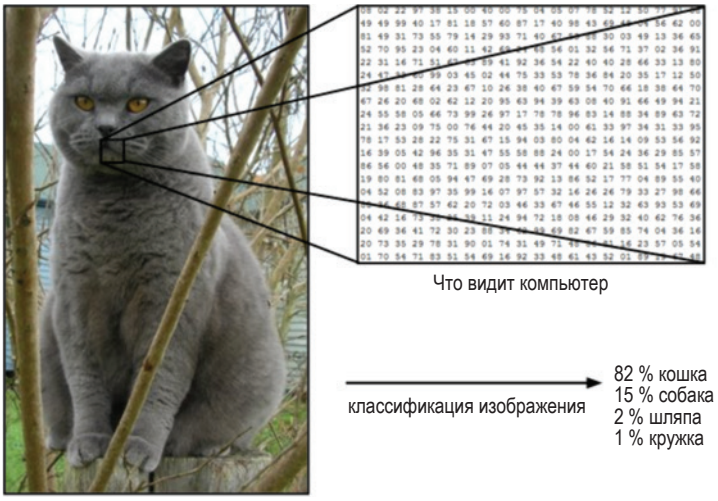


Рис. 1.2 ❖ Иллюстрация к проблеме классификации изображений. Взято с сайта https://cs231n.github.io/. Печатается с разрешения Андрея Карпатого

К счастью для нас, ботаники уже выделили 4 простых, но весьма информативных числовых признака – длину чашелистика, ширину чашелистика, длину лепестка и ширину лепестка, – с помощью которых можно различить три вида ирисов. В этом разделе мы для простоты будем использовать именно такое пространство входов куда более низкой размерности: $\mathcal{X} = \mathbb{R}^4$. **Набор данных Iris** представляет собой совокупность 150 помеченных примеров ирисов, по 50 каждого вида, описываемых этими четырьмя признаками. Он широко ис-

пользуется в качестве примера, потому что невелик и прост для понимания. (Далее в книге нам встретятся наборы данных побольше и посложнее.)

Небольшие наборы признаков часто хранят в виде **матрицы плана** размера $N \times D$, в которой строки представляют примеры, в столбцы – признаки, см. пример в табл. 1.1[1].

Таблица 1.1. Подмножество матрицы плана для ирисов

Индекс	sl	sw	pl	pw	Метка
0	5.1	3.5	1.4	0.2	Setosa
1	4.9	3.0	1.4	0.2	Setosa
	…				
50	7.0	3.2	4.7	1.4	Versicolor
	…				
149	5.9	3.0	5.1	1.8	Virginica

Признаки: длина чашелистика (sl), ширина чашелистика (sw), длина лепестка (pl), ширина лепестка (pw). В каждом классе по 50 примеров.

Набор данных Iris – пример **табличных данных**. Если размер входных данных переменный (например, последовательности слов или социальные сети), в отличие от векторов фиксированной длины, то данные обычно хранятся в формате, отличном от матрицы плана. Однако такие данные часто преобразуются в представление с признаками фиксированного размера (этот процесс называется **фичеризацией**), т. е. неявно создается матрица плана для последующей обработки. Пример такого рода мы приведем в разделе 1.5.4.1, когда будем обсуждать «представление» последовательностей в виде «мешка слов».

1.2.1.2. Разведочный анализ данных

Прежде чем применять к какой-то задаче МО, полезно провести **разведочный анализ данных** и выяснить, есть ли очевидные закономерности (их наличие может подсказать, какой метод выбрать) или очевидные проблемы (например, зашумленные метки или выбросы).

Для табличных данных с небольшим числом признаков часто строят **график парных отношений**, на котором панель (i, j) содержит диаграмму рассеяния величин i и j, а диагональные элементы (i, i) показывают маргинальную плотность величины i; дополнительно все графики могут быть окрашены цветом, кодирующим метку классу, см. пример на рис. 1.3.

Для данных более высокой размерности зачастую сначала выполняют **понижение размерности**, а затем визуализируют данные в двух или трех измерениях. Методы понижения размерности мы будем обсуждать в главе 20.

---

[1] В этой конкретной матрице плана $N = 150$ строк и $D = 4$ столбцов, т. е. она высокая и узкая, поскольку $N \gg D$. Встречаются также наборы данных (например, в геномике), в которых признаков больше, чем примеров, т. е. $D \gg N$; для них матрица плана низкая и широкая. Термин «**большие данные**» обычно означает, что N велико, а термин «**широкие данные**» – что D велико (по сравнению с N).

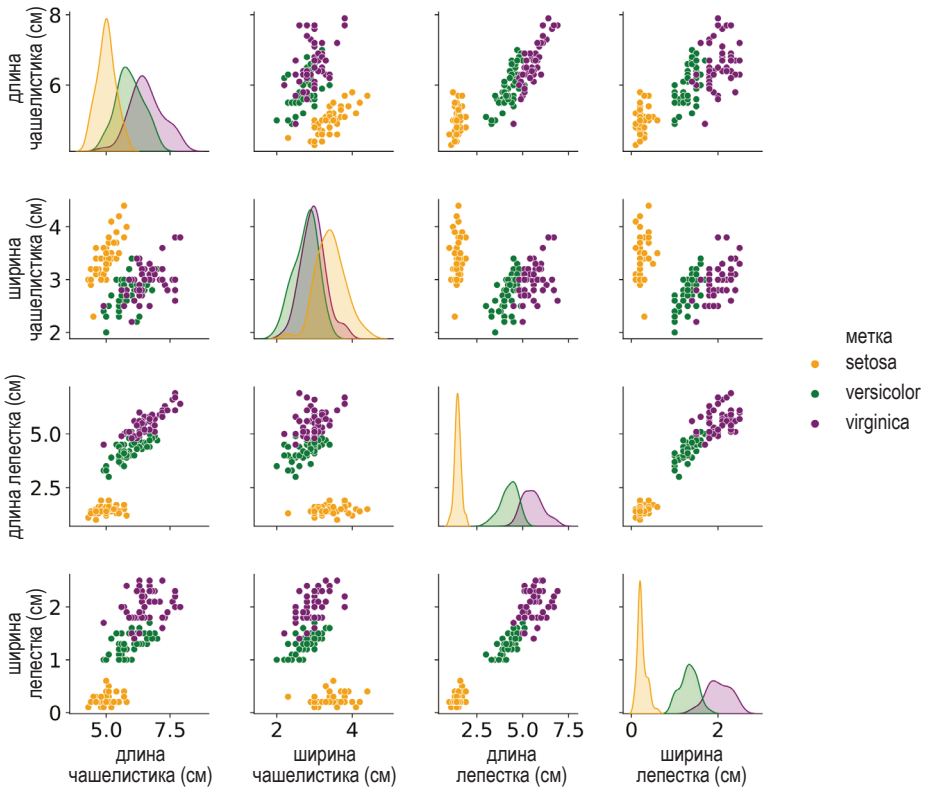


Рис. 1.3 ❖ Визуализация набора данных Iris в виде попарной диаграммы рассеяния. На главной диагонали приведены графики маргинального распределения[1] каждого признака для каждого класса. Элементы вне диагонали содержат диаграммы рассеяния всех возможных пар признаков. Построено программой по адресу figures.probml.ai/book1/1.3

1.2.1.3. Обучение классификатора

Из рис. 1.3 видно, что класс Setosa довольно легко отличить от двух остальных. Например, можно создать такое **решающее правило**:

$$f(x,\boldsymbol{\theta}) = \begin{cases} \text{Setosa, если длина лепестка} < 2.45 \\ \text{Versicolor или Virginica в противном случае} \end{cases}. \tag{1.1}$$

Это очень простой пример классификатора; мы разбили пространство входов на две области, разделенные одномерной **решающей границей** $x_{\text{длина лепестка}} = 2.45$. Точки слева от этой границы классифицируются как Setosa, а точки справа – как Versicolor или Virginica.

[1] Ранее в русскоязычной литературе термин *marginal distribution* переводился как «частное распределение», сейчас более употребителен перевод «маргинальное распределение». – *Прим. перев.*

Мы видим, что это правило абсолютно точно классифицирует примеры Setosa, но не примеры Virginica и Versicolor. Для улучшения качества мы можем рекурсивно разбить те области пространства, в которых классификатор допускает ошибки. Например, можно добавить еще одно решающее правило, применяемое к входным данным, которые не прошли первого теста, и проверить, меньше ли ширина лепестка, чем 1.75 см (в таком случае мы предсказываем класс Versicolor), или больше (и тогда мы предсказываем Virginica). Эти вложенные правила можно организовать в виде древовидной структуры, называемой **решающим деревом**, как показано на рис. 1.4a. Это дерево индуцирует двумерную **решающую поверхность**, показанную на рис. 1.4b.

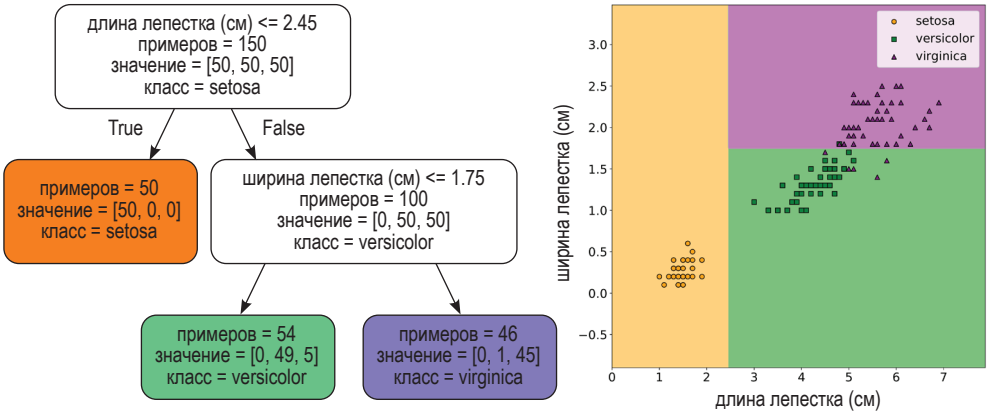


Рис. 1.4 ❖ Пример решающего дерева глубины 2 применительно к набору данных Iris с использованием только признаков «длина лепестка» и «ширина лепестка». Листовые узлы окрашены в соответствии с предсказанным классом. В каждом узле указано количество обучающих примеров, переданных от корня в этот узел; показано, сколько значений из каждого класса попадает в узел. Этот вектор счетчиков можно нормировать и получить распределение меток класса в каждом узле. Затем мы можем выбрать мажоритарный класс. На основе рис. 6.1 и 6.2 из работы [Gér19]. Построено программой по адресу figures.probml.ai/book1/1.4

Мы можем представить это дерево, сохранив для каждого внутреннего узла индекс использованного признака и соответствующее пороговое значение. Все эти **параметры** будем обозначать $\boldsymbol{\theta}$. Как выполнить обучение этих параметров, мы обсудим в разделе 18.1.

1.2.1.4. Минимизация эмпирического риска

Цель обучения с учителем – автоматическое построение моделей классификации типа показанной на рис. 1.4a, так чтобы они надежно предсказывали метки для любых подаваемых на вход данных. Часто для измерения качества модели на такой задаче применяется **частота неправильной классификации** на обучающем наборе:

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq \frac{1}{N}\sum_{n=1}^{N}\mathbb{I}(y_n \neq f(\boldsymbol{x}_n;\boldsymbol{\theta})), \tag{1.2}$$

где $\mathbb{I}(e)$ – бинарная **индикаторная функция**, которая возвращает 1, если условие e истинно, и 0 в противном случае, т. е.:

$$\mathbb{I}(e) = \begin{cases} 1, & \text{если } e \text{ равно true} \\ 0, & \text{если } e \text{ равно false} \end{cases}. \tag{1.3}$$

Здесь предполагается, что все ошибки равноценны. Однако бывают случаи, когда одни ошибки обходятся дороже, чем другие. Предположим, к примеру, что мы ищем съедобные растения в лесу и нашли какие-то ирисы. Еще предположим, что луковицы ирисов Setosa и Versicolor вкусные, а Virginica ядовитые. В таком случае можно было бы использовать асимметричную **функцию потерь** $\ell(y, \hat{y})$, показанную в табл. 1.2.

Таблица 1.2. Гипотетическая асимметричная матрица потерь для классификации набора данных Iris

		Оценка		
		Setosa	Versicolor	Virginica
Истина	Setosa	0	1	1
	Versicolor	1	0	1
	Virginica	10	10	0

Тогда можно определить **эмпирический риск** как среднюю потерю предиктора на обучающем наборе:

$$\mathcal{L}(\boldsymbol{\theta}) \triangleq \frac{1}{N}\sum_{n=1}^{N}\ell(y_n, f(\boldsymbol{x}_n;\boldsymbol{\theta})). \tag{1.4}$$

Мы видим, что частота неправильной классификации (1.2) равна эмпирическому риску, если для сравнения истинной метки с предсказанной использовать **бинарную функцию потерь** (zero-one loss):

$$\ell_{01}(y, \hat{y}) = \mathbb{I}(y \neq \hat{y}). \tag{1.5}$$

Детали см. в разделе 5.1.

Один из способов поставить задачу о **подгонке** или **обучении модели** – найти параметры, доставляющие минимум эмпирическому риску на обучающем наборе:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\,\mathcal{L}(\boldsymbol{\theta}) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\,\frac{1}{N}\sum_{n=1}^{N}\ell(y_n, f(\boldsymbol{x}_n;\boldsymbol{\theta})). \tag{1.6}$$

Такая постановка называется **минимизацией эмпирического риска**.

Однако наша истинная цель состоит в том, чтобы минимизировать ожидаемую потерю на будущих данных, которые не предъявлялись модели. То

есть мы хотим добиться **обобщаемости**, а не просто хорошего качества на обучающем наборе. Мы обсудим этот важный момент в разделе 1.2.3.

1.2.1.5. Неопределенность

> [Надлежит избегать] ложной уверенности, проистекающей из неведения вероятностной природы мира, из желания видеть черное и белое там, где правильнее было бы видеть серое.
>
> — *Иммануил Кант* в пересказе Марии Конниковой [Kon20]

Во многих случаях мы не можем точно предсказать, какой выход воспоследует из известного входа, поскольку не знаем, как устроено отображение входа в выход (это называется **эпистемической неопределенностью** или **неопределенностью модели**), и (или) вследствие внутренне присущей (неустранимой) стохастичности отображения (это называется **алеаторной неопределенностью** или **неопределенностью данных**).

Представление неопределенности в предсказании может играть важную роль в некоторых приложениях. Например, вернемся к примеру с отравленным цветком, матрица потерь которого показана в табл. 1.2. Если мы с высокой вероятностью предсказываем, что это ирис Virginica, то не должны есть такую луковицу. С другой стороны, мы могли бы заняться **сбором информации**, скажем выполнить диагностический тест, чтобы уменьшить неопределенность. Дополнительные сведения о принятии оптимальных решений в условиях неопределенности см. в разделе 5.1.

Мы можем описать неопределенность с помощью следующего **условного распределения вероятностей**:

$$p(y = c|\boldsymbol{x}; \boldsymbol{\theta}) = f_c(\boldsymbol{x}; \boldsymbol{\theta}), \tag{1.7}$$

где $f : \mathcal{X} \to [0, 1]^C$ – отображение распределения вероятностей C возможных выходных меток. Поскольку $f_c(\boldsymbol{x}; \boldsymbol{\theta})$ возвращает вероятность метки класса c, мы требуем, чтобы для каждого c было $0 \leq f_c \leq 1$ и $\sum_{c=1}^{C} f_c = 1$. Чтобы избежать этого ограничения, часто требуют вместо этого, чтобы модель возвращала ненормированные логарифмы вероятностей. Затем мы преобразуем их в вероятности с помощью **функции softmax**, определенной следующим образом:

$$\mathcal{S}(\boldsymbol{a}) \triangleq \left[\frac{e^{a_1}}{\sum_{c'=1}^{C} e^{a_{c'}}}, \ldots, \frac{e^{a_C}}{\sum_{c'=1}^{C} e^{a_{c'}}} \right]. \tag{1.8}$$

Эта функция отображает $\mathbb{R}^C$ в $[0, 1]^C$ и удовлетворяет ограничениям $0 \leq \mathcal{S}(\boldsymbol{a})_c \leq 1$ и $\sum_{c=1}^{C} \mathcal{S}(\boldsymbol{a})_c = 1$. Входные данные softmax, $\boldsymbol{a} = f(\boldsymbol{x}; \boldsymbol{\theta})$, называются **логитами**. Подробнее см. раздел 2.5.2. Таким образом, мы определяем общую модель в виде:

$$p(y = c|\boldsymbol{x}; \boldsymbol{\theta}) = \mathcal{S}_c(\boldsymbol{x}; \boldsymbol{\theta}). \tag{1.9}$$

В распространенном частном случае этого определения f – **аффинная функция** вида

$$f(\boldsymbol{x};\boldsymbol{\theta}) = b + \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} = b + w_1x_1 + w_2x_2 + \cdots + w_Dx_D, \tag{1.10}$$

где $\boldsymbol{\theta} = (b, \boldsymbol{w})$ – параметры модели. Такая модель называется **логистической регрессией** и будет подробно обсуждаться в главе 10.

В статистике параметры $\boldsymbol{w}$ обычно называются **коэффициентами регрессии** (и обозначаются β), а b – **свободным членом**. В МО параметры $\boldsymbol{w}$ называются **весами**, а b – **смещением**. Эта терминология заимствована из электротехники, где функция f рассматривается как электрическая цепь, которая принимает на входе $\boldsymbol{x}$ и возвращает $f(\boldsymbol{x})$. Входы подаются цепи по «проводам», имеющим веса $\boldsymbol{w}$. Цепь вычисляет сумму входов и прибавляет постоянное смещение b. (Это употребление термина «смещение» не следует путать со статистическим понятием смещения, обсуждаемым в разделе 4.7.6.1.)

Чтобы не загромождать обозначения, принято включать смещение b в состав весов $\boldsymbol{w}$, определив векторы $\tilde{\boldsymbol{w}} = [b, w_1, \ldots, w_D]$ и $\tilde{\boldsymbol{x}} = [1, x_1, \ldots, x_D]$, так что:

$$\tilde{\boldsymbol{w}}^{\mathsf{T}}\tilde{\boldsymbol{x}} = b + \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}. \tag{1.11}$$

Тем самым мы преобразуем аффинную функцию в **линейную функцию**. Обычно мы будем предполагать, что это уже сделано, поэтому функцию предсказания можно записать в виде:

$$f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}. \tag{1.12}$$

1.2.1.6. Оценка максимального правдоподобия

При обучении вероятностных моделей часто используют отрицательную логарифмическую вероятность в качестве функции потерь:

$$\ell(y; f(\boldsymbol{x};\boldsymbol{\theta})) = -\log p(y|f(\boldsymbol{x};\boldsymbol{\theta})). \tag{1.13}$$

Причины этого будут раскрыты в разделе 5.1.6.1, но интуиция подсказывает, что хорошей (с низкими потерями) является модель, которая назначает высокую вероятность истинному выходу y, соответствующему входу $\boldsymbol{x}$. Средняя отрицательная логарифмическая вероятность обучающего набора равна

$$\mathrm{NLL}(\boldsymbol{\theta}) = -\frac{1}{N}\sum_{n=1}^{N}\log p(y_n|f(\boldsymbol{x}_n;\boldsymbol{\theta})). \tag{1.14}$$

Эта величина и называется **отрицательным логарифмическим правдоподобием**. Минимизировав ее, мы найдем **оценку максимального правдоподобия** (maximum likelihood estimate – **MLE**):

$$\hat{\boldsymbol{\theta}}_{\mathrm{mle}} = \operatorname*{argmin}_{\boldsymbol{\theta}} \mathrm{NLL}(\boldsymbol{\theta}). \tag{1.15}$$

Ниже мы увидим, что это очень распространенный способ подгонки моделей к данным.

1.2.2. Регрессия

Теперь предположим, что требуется предсказать вещественную величину $y \in \mathbb{R}$, а не метку класса $y \in \{1, \ldots, C\}$; такая задача называется **регрессией**. Так, в случае набора Iris y может быть степенью токсичности луковицы цветка или средней высотой растения.

Регрессия очень похожа на классификацию. Но, поскольку выходом является вещественное значение, нам нужна другая функция потерь. Для регрессии чаще всего используют **квадратичную потерю**, или **ℓ_2-потерю**:

$$\ell_2(y; \hat{y}) = (y - \hat{y})^2. \tag{1.16}$$

Эта функция штрафует за большие **невязки** $y - \hat{y}$ сильнее, чем за малые[1]. Эмпирический риск при использовании квадратичной потери равен **среднеквадратической ошибке** (**СКО**, англ. **MSE**):

$$\text{MSE}(\boldsymbol{\theta}) = \frac{1}{N}\sum_{n=1}^{N}(y_n - f(\boldsymbol{x}_n; \boldsymbol{\theta}))^2. \tag{1.17}$$

Вспоминая обсуждение в разделе 1.2.1.5, мы должны также смоделировать неопределенность предсказания. В задачах регрессии принято предполагать, что распределение выходов **гауссово**, или **нормальное**. В разделе 2.6 мы объясним, что это распределение определено формулой

$$\mathcal{N}(y|\mu, \sigma^2) \triangleq \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(y-\mu)^2}, \tag{1.18}$$

где μ – среднее, σ^2 – дисперсия, а $\sqrt{2\pi\sigma^2}$ – нормировочная постоянная, которая необходима, чтоб интеграл плотности был равен 1. В контексте регрессии среднее может зависеть от входов: $\mu = f(\boldsymbol{x}_n; \boldsymbol{\theta})$. Поэтому мы получаем такое условное распределение вероятностей:

$$p(y|\boldsymbol{x}; \boldsymbol{\theta}) = \mathcal{N}(y|f(\boldsymbol{x}; \boldsymbol{\theta}); \sigma^2). \tag{1.19}$$

Если предположить, что дисперсия σ^2 фиксирована (для простоты), то отрицательное логарифмическое правдоподобие принимает вид:

$$\text{NLL}(\boldsymbol{\theta}) = -\sum_{n=1}^{N}\log\left[\left(\frac{1}{2\pi\sigma^2}\right)^{\frac{1}{2}}\exp\left(-\frac{1}{2\sigma^2}(y_n - f(\boldsymbol{x}_n; \boldsymbol{\theta}))^2\right)\right] \tag{1.20}$$

$$= \frac{N}{2\sigma^2}\text{MSE}(\boldsymbol{\theta}) + \text{const}. \tag{1.21}$$

---

[1] Если в данных имеются выбросы, то квадратичный штраф может оказаться слишком суровым. В таких случаях лучше использовать более **робастную** ℓ_1-потерю. Подробнее см. раздел 11.6.

Мы видим, что NLL пропорциональна MSE. Поэтому вычисление оценки максимального правдоподобия параметров сводится к минимизации квадратичной ошибки, что представляется разумным подходом к подгонке модели.

1.2.2.1. Линейная регрессия

В качестве примера модели регрессии рассмотрим одномерные данные на рис. 1.5a. Их можно аппроксимировать простой моделью **линейной регрессии** вида

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = b + wx, \tag{1.22}$$

где w – **угловой коэффициент**, b – **смещение**, а $\boldsymbol{\theta} = (w, b)$ – параметры модели. Варьируя $\boldsymbol{\theta}$, мы можем уменьшать сумму квадратов ошибок, представленных вертикальными отрезками на рис. 1.5b, и в итоге найдем решение с наименьшей СКО:

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \operatorname{MSE}(\boldsymbol{\theta}). \tag{1.23}$$

Детали см. в разделе 11.2.2.1.

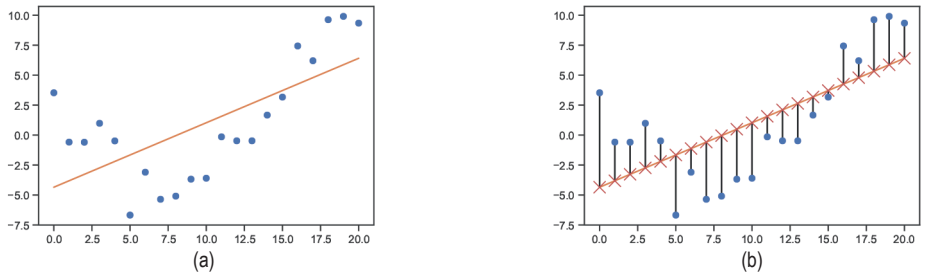


Рис. 1.5 ❖ (a) Линейная регрессия для одномерных данных. (b) Вертикальные отрезки обозначают невязки между наблюдаемым (синий кружочек) и предсказанным (красный крестик) выходным значением. Цель регрессии методом наименьших квадратов – выбрать линию, которая минимизирует сумму квадратов невязок. Сгенерировано программой по адресу figures.probml.ai/book1/1.5

При наличии нескольких входных признаков мы можем написать

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = b + w_1 x_1 + \cdots + w_D x_D = b + \boldsymbol{w}^{\mathsf{T}} \boldsymbol{x}, \tag{1.24}$$

где $\boldsymbol{\theta} = (\boldsymbol{w}, b)$. Это называется **многофакторной линейной регрессией**.

Например, рассмотрим задачу предсказания температуры как функции двумерного местоположения в комнате. На рис. 1.6a показаны результаты линейной модели вида

$$f(\boldsymbol{x}; \boldsymbol{\theta}) = b + w_1 x_1 + w_2 x_2. \tag{1.25}$$

Мы можем обобщить эту модель на $D > 2$ входных признаков (скажем, добавить время суток), но тогда ее будет труднее визуализировать.

1.2.2.2. Полиномиальная регрессия

Линейная модель на рис. 1.5a, очевидно, не очень хорошо аппроксимирует данные. Мы можем улучшить дело, воспользовавшись **моделью полиномиальной регрессии** степени D. Она имеет вид $f(x; \boldsymbol{w}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}(x)$, где $\boldsymbol{\phi}(x)$ – вектор признаков, построенный по входным данным и имеющий вид

$$\boldsymbol{\phi}(x) = [1; x; x^2; ...; x^D]. \tag{1.26}$$

Это простой пример **предобработки признаков**, или **конструирования признаков**. На рис. 1.7a видно, что при $D = 2$ результаты аппроксимации гораздо лучше. Можно и дальше увеличивать D, а вместе с ним и количество параметров модели, вплоть до $D = N - 1$, когда число параметров сравняется с числом точек и интерполяция будет идеальной. В такой модели СКО будет равно 0, как показано на рис. 1.7c. Однако интуитивно понятно, что результирующая функция вряд ли окажется хорошим предиктором для будущих данных, потому что она слишком «заковыристая». Мы обсудим этот вопрос подробнее в разделе 1.2.3.

Мы можем также применить полиномиальную регрессию к многомерным входным данным. Например, на рис. 1.6b показаны предсказания модели температуры, когда входные данные аппроксимируются квадратичной функцией:

$$f(\boldsymbol{x}; \boldsymbol{w}) = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_2^2. \tag{1.27}$$

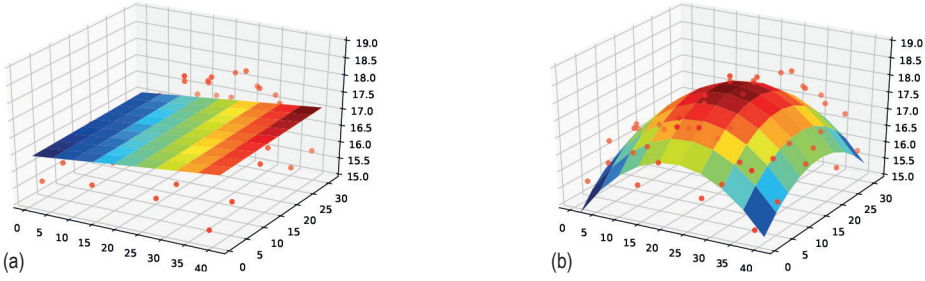


Рис. 1.6 ❖ Линейная и полиномиальная регрессия применительно к двумерным данным. По вертикальной оси отложена температура, а по горизонтальным – местоположение в комнате. Данные были собраны с помощью беспроводных однокристальных датчиков в лаборатории Intel в Беркли, штат Калифорния (используются с разрешения Ромейна Тибо). (a) Аппроксимирующая плоскость вида $\hat{f}(x) = w_0 + w_1x_1 + w_2x_2$. (b) Данные о температуре аппроксимированы квадратичной функцией вида $\hat{f}(x) = w_0 + w_1x_1 + w_2x_2 + w_3x_1^2 + w_4x_1^2$. Построено программой по адресу figures.probml.ai/book1/1.6

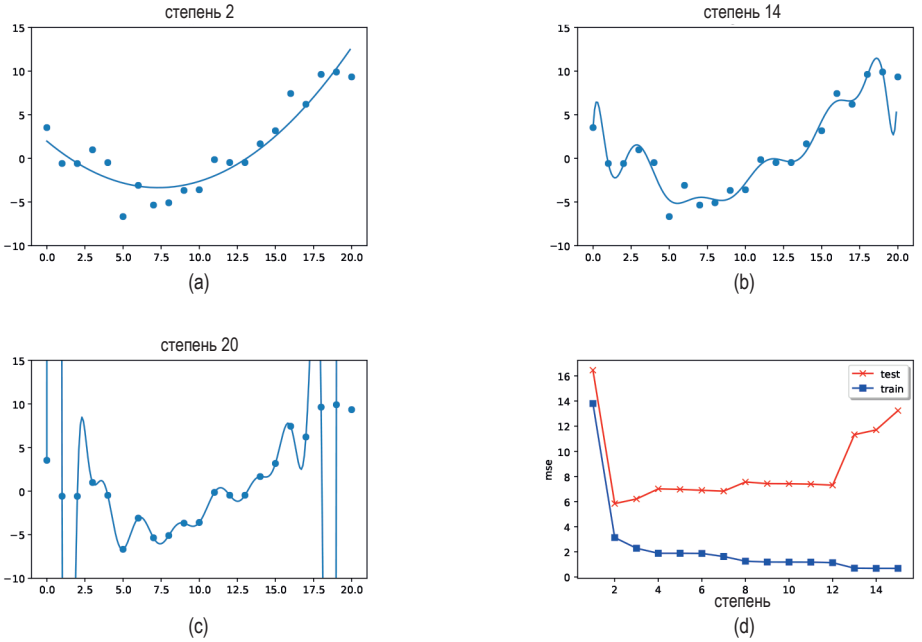


Рис. 1.7 ❖ (a–c) Аппроксимация 21 точки полиномами степени 2, 14 и 20 (данные те же, что на рис. 1.5). (d) Зависимость СКО от степени. Построено программой по адресу figures.probml.ai/book1/1.7

Квадратичная модель лучше аппроксимирует данные, чем линейная, показанная на рис. 1.6a, потому что улавливает тот факт, что в центре комнаты теплее. Можно также добавить перекрестные произведения типа $x_1 x_2$, чтобы уловить эффекты взаимодействия. Детали см. в разделе 1.5.3.2.

Отметим, что во всех перечисленных выше моделях функция-предиктор линейно зависит от параметров $\boldsymbol{w}$, хотя и нелинейно от входных данных $\boldsymbol{x}$. Это важно, потому что линейная модель индуцирует среднеквадратическую функцию потерь MSE($\boldsymbol{\theta}$), имеющую единственный глобальный оптимум, как будет объяснено в разделе 11.2.2.1.

1.2.2.3. Глубокие нейронные сети

В разделе 1.2.2.2 мы вручную задали преобразование входных признаков – полиномиальное разложение $\boldsymbol{\phi}(\boldsymbol{x}) = [1, x_1, x_2, x_1^2, x_2^2, \ldots]$. Можно создать гораздо более мощные модели, обучившись производить такое нелинейное **выделение признаков** автоматически. Если считать, что $\boldsymbol{\phi}(\boldsymbol{x})$ имеет собственный набор параметров, скажем $\mathbf{V}$, то полная модель будет иметь вид:

$$f(\boldsymbol{x}; \boldsymbol{w}; \mathbf{V}) = \boldsymbol{w}^{\mathsf{T}} \boldsymbol{\phi}(\boldsymbol{x}; \mathbf{V}). \tag{1.28}$$

Мы можем рекурсивно разложить функцию выделения признаков $\boldsymbol{\phi}(\boldsymbol{x}; \mathbf{V})$ в композицию более простых функций. В результате получается модель, состоящая из L вложенных функций:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = f_L(f_{L-1}(\cdots(f_1(\boldsymbol{x}))\cdots)), \tag{1.29}$$

где $f_\ell(\boldsymbol{x}) = f(\boldsymbol{x};\boldsymbol{\theta}_\ell)$ – функция слоя ℓ. Последний слой линейный и имеет вид $f_L(\boldsymbol{x}) = \boldsymbol{w}^\mathsf{T} f_{1:L-1}(\boldsymbol{x})$, где $f_{1:L-1}(\boldsymbol{x})$ – обученная функция выделения признаков (экстрактор). Эта идея – ключ к **глубоким нейронным сетям** (**ГНС**), у которых есть такие широко известные варианты, как **сверточные нейронные сети** (**СНС**) для изображений и **рекуррентные нейронные сети** (**РНС**) для последовательностей. Подробнее см. часть III.

1.2.3. Переобучение и обобщаемость

Мы можем переписать эмпирический риск в формуле (1.4) в эквивалентной, но более удобной форме:

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{D}_{\text{train}}) = \frac{1}{|\mathcal{D}_{\text{train}}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}_{\text{train}}} \ell(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta})), \tag{1.30}$$

где $|\mathcal{D}_{\text{train}}|$ – размер обучающего набора $\mathcal{D}_{\text{train}}$. Эта форма полезна, потому что из нее явно видно, на каком наборе данных вычисляется функция потерь.

Если модель достаточно гибкая, то можно свести потери на обучающем наборе к нулю (в предположении, что метки незашумленные), просто запомнив правильный выход для каждого входа. Так, модель на рис. 1.7c идеально интерполирует обучающие данные (не считая одной точки справа). Но нас-то интересует точность предсказания на новых данных, которые могут и не быть частью обучающего набора. Если модель точно аппроксимирует обучающие данные, но при этом слишком сложна, то говорят, что она **переобучена**.

Чтобы выяснить, является ли модель переобученной, предположим (пока), что имеется доступ к истинному (но неизвестному) распределению $p^*(\boldsymbol{x}, \boldsymbol{y})$, использованному для порождения обучающего набора. Тогда вместо вычисления эмпирического риска мы вычисляем теоретически ожидаемую потерю, или **риск на генеральной совокупности:**

$$\mathcal{L}(\boldsymbol{\theta};p^*) \triangleq \mathbb{E}_{p^*(\boldsymbol{x},\boldsymbol{y})}[\ell(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta}))]. \tag{1.31}$$

Разность $\mathcal{L}(\boldsymbol{\theta};\, p^*) - \mathcal{L}(\boldsymbol{\theta};\, \mathcal{D}_{\text{train}})$ называется **разрывом обобщения**. Если разрыв обобщения для модели велик (т. е. эмпирический риск низкий, а риск на генеральной совокупности высокий), то можно заподозрить переобучение. На практике распределение p^* неизвестно. Однако мы можем разбить имеющиеся данные на два подмножества: обучающий набор и **тестовый набор**. И аппроксимировать риск на генеральной совокупности **риском на тестовом наборе**:

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{D}_{\text{test}}) \triangleq \frac{1}{|\mathcal{D}_{\text{test}}|}\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}_{\text{test}}} \ell(\boldsymbol{y}, f(\boldsymbol{x};\boldsymbol{\theta})). \tag{1.32}$$

Так, на рис. 1.7d построен график ошибки на обучающем и на тестовом наборе для полиномиальной регрессии в виде функции от степени D. Мы видим, что ошибка на обучающем наборе стремится к 0 при усложнении модели. Однако ошибка на тестовом наборе имеет характерную U-образную форму: в левой части, где $D = 1$, модель недообучена, а справа, где $D \gg 1$, переобучена. А вот при $D = 2$ сложность модели как раз такая, «как надо».

Как правильно выбрать сложность модели? Если для оценивания различных моделей использовать обучающий набор, то мы всегда будем выбирать самую сложную модель, потому что у нее наибольшее число **степеней свободы**, а значит, наименьшая потеря. Поэтому выбирать следует модель с минимальной потерей на тестовом наборе.

На практике данные следует разбивать на три множества: обучающий набор, тестовый и **контрольный набор**; последний используется для выбора модели, а тестовый только для оценки качества на будущих данных (риска на генеральной совокупности). То есть тестовый набор не используется ни для обучения модели, ни для ее выбора. Дальнейшие детали см. в разделе 4.5.4.

1.2.4. Теорема об отсутствии бесплатных завтраков

Все модели неверны, но некоторые из них полезны.
— Джордж Бокс [BD87, стр. 424][1]

В литературе описано множество самых разных моделей, поэтому естественно спросить, какая их них наилучшая. К сожалению, не существует одной модели, которая была бы оптимальна для всех задач, – иногда этот факт называют **теоремой об отсутствии бесплатных завтраков** [Wol96]. Причина в том, что набор допущений (иногда называемый **индуктивным смещением**), который хорошо работает в одной предметной области, может давать никуда не годные результаты в другой. При выборе подходящей модели лучше всего опираться на знание предметной области и (или) на метод проб и ошибок (т. е. применять такие способы выбора модели, как перекрестная проверка (раздел 4.5.4) или байесовские методы (раздел 5.2.2)). Поэтому так важно иметь в своем арсенале много моделей и алгоритмических методов.

1.3. Обучение без учителя

В обучении с учителем предполагается, что с каждым входным примером $\boldsymbol{x}$ в обучающем наборе ассоциировано множество выходных меток $\boldsymbol{y}$, а наша цель – обучиться отображению входа в выход. Хотя это полезно, а иногда трудно, по сути дела, обучение с учителем – всего лишь «напыщенная аппроксимация кривой» [Pea18].

[1] Джордж Бокс – профессор статистики Висконсинского университета, ныне на пенсии.

Гораздо более интересная задача – попытаться «извлечь смысл» из данных, а не просто обучить отображение. То есть у нас есть только наблюдаемые «входы» $\mathcal{D} = \{\boldsymbol{x}_n : n = 1 : N\}$, но никаких соответствующих им «выходов» $\boldsymbol{y}_n$. Такая постановка задачи называется **обучением без учителя**.

С вероятностной точки зрения, задачу обучения без учителя можно рассматривать как обучение безусловной модели вида $p(\boldsymbol{x})$, которая умеет порождать новые данные $\boldsymbol{x}$, тогда как обучение с учителем – это обучение условной модели $p(\boldsymbol{y}|\boldsymbol{x})$, задающей распределение выходов при условии входов[1].

В обучении без учителя нет нужды собирать большие помеченные наборы данных для обучения, что может оказаться делом долгим и дорогостоящим (представьте, что вы просите врачей пометить медицинские изображения).

Нет также нужды обучаться разбиению мира на категории, зачастую произвольные. Подумайте, например, о задаче пометить на видео действие, скажем «питье» или «прихлебывание». Когда оно начинается: когда человек поднимает стакан или когда стакан касается губ? Люди зачастую не могут прийти к единой точке зрения по таким вопросам [Idr+17], а значит, задача поставлена некорректно. Поэтому странно было бы ожидать, что машина сможет обучиться такому отображению[2].

Наконец, обучение без учителя вынуждает модель «объяснять» входы высокой размерности, а не только выходы низкой размерности. Это позволяет обучать более развитые модели «устройства мира».

Джеффри Хинтон, знаменитый профессор МО из Торонтского университета, писал:

> Когда мы учимся видеть, никто не подсказывает нам правильные ответы – мы просто смотрим. Бывает, мама говорит «это собака», но информации в этих словах очень мало. Хорошо, если таким способом удается получить хоть несколько битов информации – пусть даже один бит в секунду. В зрительной системе мозга порядка 10^{14} нейронных связей. А наша жизнь продолжается всего 10^9 с. Так что обучаться со скоростью один бит в секунду бесполезно. Нужно быстрее, порядка 10^5 бит в секунду. И есть только одно место, где можно добыть так много информации: сами входные данные.
>
> — *Джеффри Хинтон* (цитируется по [Gor06])

1.3.1. Кластеризация

Простой пример обучения без учителя – проблема нахождения **кластеров** в данных. Цель – разбить множество входов на области, содержащие «похо-

---

[1] У статистиков принято использовать букву $\boldsymbol{x}$ для обозначения экзогенных переменных, которые не моделируются, а просто являются входными данными. Поэтому безусловная модель обозначалась бы $p(\boldsymbol{y})$, а не $p(\boldsymbol{x})$.

[2] Более разумный подход – попытаться уловить распределение вероятностей меток, проставленных «толпой» аннотаторов (см., например, [Dum+18; Aro+19]). При этом учитывается тот факт, что данному входу может соответствовать несколько «правильных» меток – просто в силу неоднозначности задачи.

жие» точки. Рассмотрим двумерный вариант набора данных Iris. На рис. 1.8a показаны точки без меток класса. Интуитивно понятно, что в данных имеется по меньшей мере два кластера, в левом нижнем и в правом верхнем углу. Кроме того, если предположить, что «хороший» кластер должен быть относительно компактным, то точки в правом верхнем углу хочется разбить на два подкластера (по крайней мере). Результирующее разбиение на три кластера показано на рис. 1.8b. (Отметим, что никто не говорит, сколько должно быть кластеров; мы сами должны выбрать компромисс между сложностью модели и качеством аппроксимации ею данных. Как это сделать, обсуждается в разделе 21.3.7.)

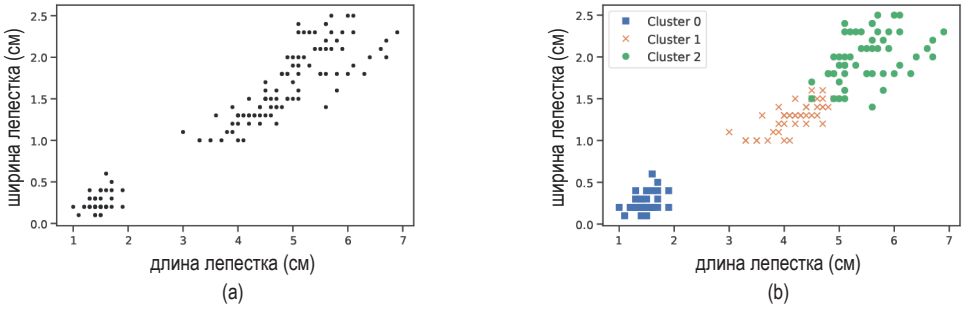


Рис. 1.8 ❖ (a) Диаграмма рассеяния признаков, относящихся к лепесткам, в наборе данных об ирисах. (b) Результат кластеризации без учителя при $K = 3$. Построено программой по адресу figures.probml.ai/book1/1.8

1.3.2. Обнаружение латентных «факторов изменчивости»

При работе с данными высокой размерности часто бывает полезно понизить размерность путем проецирования на подпространство меньшей размерности, которое улавливает основные особенности данных. Один из подходов к этой проблеме – предположить, что все наблюдаемые многомерные выходы $\boldsymbol{x}_n \in \mathbb{R}^D$ порождены множеством скрытых, т. е. ненаблюдаемых **латентных факторов** низкой размерности $\boldsymbol{z}_n \in \mathbb{R}^K$. Мы можем наглядно представить модель в виде $\boldsymbol{z}_n \rightarrow \boldsymbol{x}_n$, где стрелка обозначает причинно-следственную связь. Поскольку латентные факторы $\boldsymbol{z}_n$ нам неизвестны, часто предполагается простая модель априорной вероятности для $p(\boldsymbol{z}_n)$, например гауссова, согласно которой каждый фактор является случайным K-мерным вектором. Если данные вещественные, то можно также использовать гауссово правдоподобие.

Простейший пример – использование линейной модели $p(\boldsymbol{x}_n|\boldsymbol{z}_n; \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}_n|\mathbf{W}\boldsymbol{z}_n + \boldsymbol{\mu}, \boldsymbol{\Sigma})$. Результирующая модель называется **факторным анализом** (ФА). Он похож на линейную регрессию с тем отличием, что мы наблюдаем только выходы $\boldsymbol{x}_n$, а не входы $\boldsymbol{z}_n$. В частном случае, когда $\boldsymbol{\Sigma} = \sigma^2\mathbf{I}$, это сводится к модели вероятностного **метода главных компонент** (principal components analysis — **PCA**), который мы будем рассматривать в разделе 20.1.

На рис. 1.9 показано, как этот метод позволяет найти двумерное линейное подпространство, будучи применен к простым трехмерным данным.

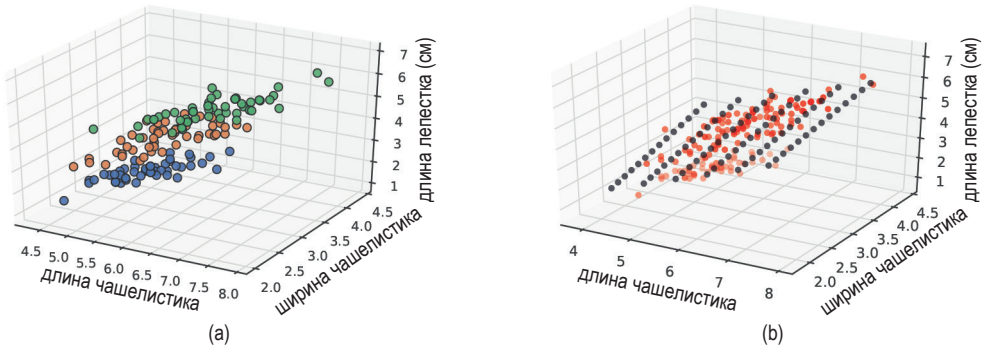


Рис. 1.9 ❖ (a) Диаграмма рассеяния признаков данных об ирисах (первые три признака). Цвет точек соответствует классам. (b) Мы аппроксимируем трехмерные данные двумерным линейным подпространством, применяя метод PCA. Метки классов игнорируются. Красные точки – оригинальные данные, черные точки сгенерированы по модели с использованием $\tilde{\boldsymbol{x}} = \mathbf{W}\boldsymbol{z} + \boldsymbol{\mu}$, где $\boldsymbol{z}$ – латентные точки на выведенном двумерном линейном многообразии. Построено программой по адресу figures.probml.ai/book1/1.9

Конечно, предположение о линейности отображения $\boldsymbol{z}_n$ в $\boldsymbol{x}_n$ чрезмерно ограничительно. Однако мы можем создать нелинейные обобщения, определив $p(\boldsymbol{x}_n|\boldsymbol{z}_n; \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}_n|f(\boldsymbol{z}_n; \boldsymbol{\theta}), \sigma^2\mathbf{I})$, где $f(\boldsymbol{z}; \boldsymbol{\theta})$ – нелинейная модель, например глубокая нейронная сеть. Обучить такую модель (т. е. оценить параметры $\boldsymbol{\theta}$) оказывается гораздо труднее, потому что необходимо выводить как входы нейронной сети, так и параметры модели. Но существуют различные приближенные методы, например вариационные автокодировщики (см. раздел 20.3.5).

1.3.3. Самостоятельное обучение

Недавно приобрел популярность новый подход к обучению без учителя – **самостоятельное обучение** (self-supervised learning). В этом случае мы создаем замещающие задачи обучения с учителем по непомеченным данным. Например, можно попробовать обучиться предсказанию цветного изображения по полутоновому или замаскировать некоторые слова в предложении и попытаться предсказать их по окружающему контексту. Мы надеемся, что получившийся в результате предиктор $\hat{\boldsymbol{x}}_1 = f(\boldsymbol{x}_2; \boldsymbol{\theta})$, где $\boldsymbol{x}_2$ – наблюдаемый вход, а $\hat{\boldsymbol{x}}_1$ – предсказанный выход, сможет обучиться полезным признакам на основе данных, а затем использовать их в последующих стандартных задачах обучения с учителем. Тем самым мы обходим трудную проблему вывода «истинных латентных факторов» $\boldsymbol{z}$, стоящих за наблюдаемыми данными, а вместо этого полагаемся на стандартные методы обучения с учителем. Мы подробнее обсудим этот подход в разделе 19.2.

1.3.4. Оценка обучения без учителя

Хотя идея обучения без учителя кажется весьма соблазнительной, очень трудно оценить качество такого метода обучения, потому что результат не с чем сравнивать [TOB16]. Наиболее распространенный способ оценки моделей, обученных без учителя, – измерение вероятности, назначенной моделью не предъявлявшимся ранее тестовым примерам. Это можно сделать, вычислив (безусловное) отрицательное логарифмическое правдоподобие данных:

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{D}) = -\frac{1}{|\mathcal{D}|}\sum_{\boldsymbol{x}\in\mathcal{D}} \log p(\boldsymbol{x}|\boldsymbol{\theta}). \tag{1.33}$$

Тем самым мы сводим проблему обучения без учителя к проблеме **оценивания плотности**. Идея в том, что хорошую модель не должны «удивлять» примеры фактических данных (т. е. она назначает им высокую вероятность). Кроме того, сумма вероятностей должна быть равна 1.0, а значит, если модель назначает высокую вероятность тем областям пространства данных, из которых взяты примеры, то областям, откуда нет данных, неявно назначается низкая вероятность. Таким образом, модуль обучилась улавливать **типичные паттерны** в данных. Это можно использовать в алгоритме **сжатия данных**.

К сожалению, оценить плотность трудно, особенно если размерность велика. Кроме того, модель, которая назначает высокую вероятность данным, может не обучиться полезным паттернам (ведь она могла бы просто запомнить все обучающие примеры).

Альтернативный способ оценивания – использовать представление, полученное в результате обучения без учителя, в качестве признаков или подать на вход следующему далее методу обучения с учителем. Если метод обучения без учителя выявил полезные паттерны, то, возможно, ими удастся воспользоваться для обучения с учителем на гораздо меньшем объеме помеченных данных, чем при работе с оригинальными признаками. Например, в разделе 1.2.1.1 мы видели, что четыре вручную определенных признака ирисов содержат большую часть информации, необходимой для классификации. И таким образом сумели обучить почти идеальный классификатор всего на 150 примерах. Если бы на вход подавались исходные пиксели, то для достижения сравнимого качества потребовалось бы куда больше примеров (см. раздел 14.1). Следовательно, мы сможем повысить **выборочную эффективность** обучения (т. е. уменьшить количество помеченных примеров, необходимых для получения приемлемого качества), если предварительно обучимся хорошему представлению.

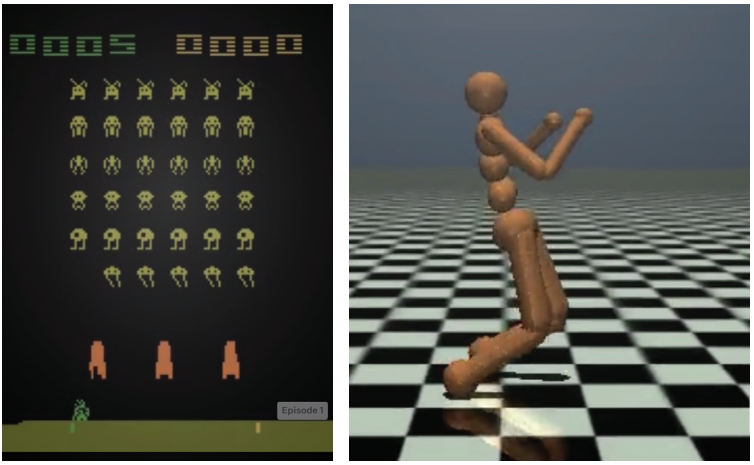
Повышенная выборочная эффективность – полезная метрика оценки, но во многих приложениях, особенно научных, цель обучения без учителя – *понять данные*, а не улучшить качество на какой-то задаче предсказания. Для этого нужны модели, **допускающие интерпретацию**, но при этом способные генерировать или «объяснять» большинство наблюдаемых в данных паттернов. Перефразируя Платона, можно сказать, что цель состоит в том, чтобы выяс-

нить, как «природа делится в местах сочленений»[1]. Конечно, чтобы оценить, правильно ли мы распознали истинную структуру, стоящую за некоторым набором данных, зачастую необходимо ставить эксперименты и, стало быть, взаимодействовать с миром. Мы продолжим обсуждение этой темы в разделе 1.4.

1.4. Обучение с подкреплением

Помимо обучения с учителем и без учителя, есть еще и третий вид МО – **обучение с подкреплением** (**ОП**, англ. **RL**). В проблемах этого класса система или **агент** должна обучиться взаимодействию со своим окружением. Это можно представить в виде **политики** $\boldsymbol{a} = \pi(\boldsymbol{x})$, определяющей, какое действие предпринять в ответ на любой возможный вход $\boldsymbol{x}$ (действие выводится из состояния окружающей среды).

Например, рассмотрим агента, который обучается играть в видеоигру, скажем Atari Space Invaders (см. рис. 1.10a). В этом случае входом $\boldsymbol{x}$ является изображение (или последовательность прошлых изображений), а выходом – направление движения (влево или вправо) и признак пуска ракеты. В качестве более сложного примера рассмотрим робота, обучающегося ходить (см. рис. 1.10b). Здесь вход $\boldsymbol{x}$ – множество положений и углов сочленения для всех конечностей, а выход – множество сигналов, подаваемых на приводы или электрические двигатели.



(a) (b)

Рис. 1.10 ❖ Пример некоторых задач управления. (a) Игра Space Invaders компании Atari. Взято с https://gym.openai.com/envs/SpaceInvaders-v0/. (b) Управление роботом-гуманоидом в эмуляторе MuJuCo; цель – добиться максимальной скорости ходьбы без падений. Взято с https://gym.openai.com/envs/Humanoid-v2/

[1] Отсылка к следующему месту из диалога Платона «Федр»: «Другой [род] состоит в уменьи делить предмет на виды – и делить, как водится, почленно, так чтобы, подобно плохому повару, не раздробить ни одной части». – *Прим. перев.*

Отличие от обучения с учителем заключается в том, что системе никто не говорит, какое действие наилучшее (т. е. какой выход следует сгенерировать для данного входа). Вместо этого система время от времени получает сигнал **вознаграждения** (или наказания) в ответ на предпринятые ей действия. Это скорее напоминает **обучение с критиком**, который периодически поднимает или опускает большой палец, а не с учителем, который на каждом шаге говорит, что нужно делать.

В последнее время популярность ОП заметно выросла благодаря широкой применимости (ведь в качестве сигнала вознаграждения, который агент пытается оптимизировать, можно выбирать любую интересующую метрику). Но по разным причинам заставить ОП работать труднее, чем в случае обучения с учителем или без него. Основная трудность заключается в том, что сигнал вознаграждения может подаваться лишь изредка (например, только когда агент в конечном итоге достигает желаемого состояния), и даже в этих случаях может быть неясно, какое из многочисленных действий привело к успеху. (Примером могут служить шахматы, когда единственный сигнал – выигрыш или проигрыш – подается в конце партии.)

Чтобы компенсировать скудость информации, поступающей от сигнала вознаграждения, часто используют другие источники информации, например проводимые экспертами демонстрации, которые можно использовать, как в обучении с учителем, или неразмеченные данные, которые могут использоваться системой обучения без учителя, чтобы выявить структуру окружающей среды. Благодаря таким уловкам становится реально обучиться на ограниченном числе испытаний (раундов взаимодействия с окружающей средой). Как заметил Ян Лекун в лекции на конференции NIPS[1] в 2016 году, «если представить интеллект в виде торта, то обучение без учителя будет шоколадной прослойкой, обучение с учителем – глазурью, а обучение с подкреплением – вишенкой». Эта мысль иллюстрируется на рис. 1.11.

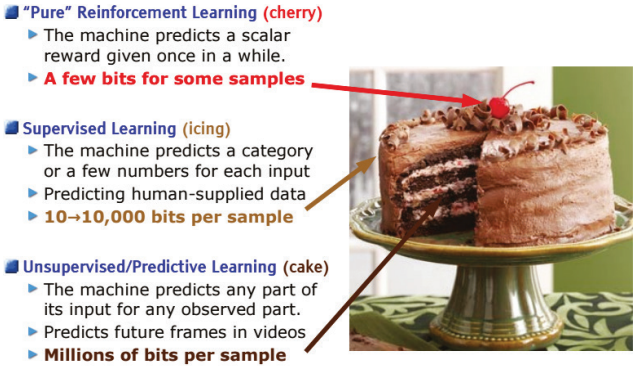


Рис. 1.11 ❖ Три типа машинного обучения, представленные в виде слоев шоколадного торта. Этот рисунок (взятый с https://bit.ly/2m65Vs1) был использован в лекции Яна Лекуна на конференции NIPS'16 и печатается с его разрешения

[1] NIPS означает Neural Information Processing Systems (Системы обработки нейронной информации). Это одна из самых авторитетных конференций по машинному обучению. Недавно она была переименована в **NeurIPS**.

Дополнительную информацию об ОП можно найти во втором томе этой книги, [Mur22].

1.5. Данные

Предмет машинного обучения – построение моделей данных с помощью различных алгоритмов. Хотя мы акцентируем внимание на аспектах моделирования и алгоритмов, необходимо отметить, что природа и качество обучающих данных также играют важную роль в успехе любой обученной модели.

В этом разделе мы кратко опишем некоторые используемые в этой книге наборы данных, содержащие изображения и текст. Мы также немного поговорим о предобработке данных.

1.5.1. Некоторые широко известные наборы изображений

В этом разделе приводится краткое обсуждение используемых в книге наборов данных, содержащих изображения.

1.5.1.1. Небольшие наборы изображений

Один из самых простых и широко распространенных наборов данных – **MNIST** [LeC+98; YB19][1]. Этот набор содержит 60 000 обучающих и 10 000 тестовых полутоновых изображений рукописных цифр из 10 категорий, каждое размером 28×28 пикселей. Каждый пиксель представлен целым числом в диапазоне {0, 1, ..., 255}; обычно они приводятся к диапазону [0, 1], чтобы представить яркость пикселей. Дополнительно можно преобразовать изображение в бинарное, воспользовавшись методом бинаризации (см. рис. 1.12a).

Набор MNIST настолько широко используется в сообществе МО, что знаменитый ученый Джеффри Хинтон назвал его «дрозофилой машинного обучения», поскольку если некоторый метод не работает на MNIST, то вряд ли он будет хорошо работать на более трудных наборах данных. Однако в наши дни классификация MNIST считается «слишком легкой» задачей, поскольку различить большинство пар цифр можно всего по одному пикселю. Были предложены различные расширения.

В работе [Coh+17] предложен набор **EMNIST** (extended MNIST), включающий также строчные и заглавные буквы (см. рис. 1.12b). Этот набор данных гораздо труднее MNIST, поскольку классов 62 и некоторые из них неоднозначны (например, сравните цифру 1 со строчной буквой l).

---

[1] Акроним MNIST означает Modified National Institute of Standards. Слово «modified» употребляется, потому что изображения были подвергнуты предварительной обработке с целью поместить цифры примерно в центр.

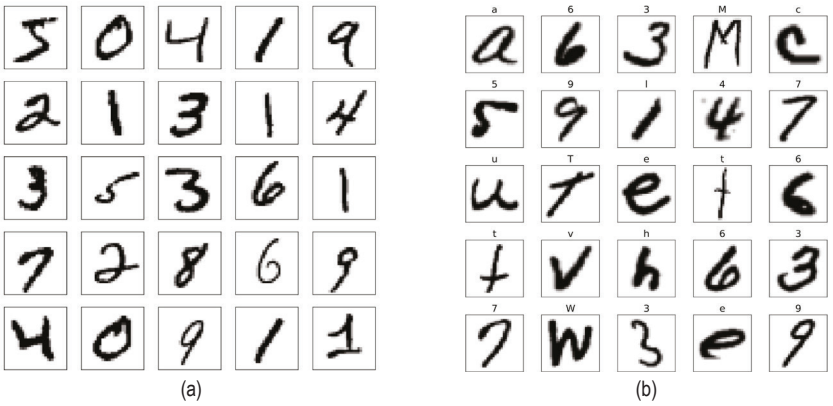


Рис. 1.12 ❖ (a) Визуализация набора данных MNIST. Размер каждого изображения 28×28. Всего имеется 60k обучающих и 10k тестовых примеров. Показаны первые 25 изображений из обучающего набора. Построено программой по адресу figures.probml.ai/book1/1.12. (b) Визуализация набора данных EMNIST. Имеется 697 932 обучающих и 116 323 тестовых примеров, размер каждого 28×28. Имеется 62 класса (a–z, A–Z, 0–9). Показаны первые 25 изображений из обучающего набора. Построено программой по адресу figures.probml.ai/book1/1.12

В работе [XRV17] предложен набор **Fashion-MNIST**. Его размер и форма изображений такие же, как в MNIST, но изображены предметы одежды, а не рукописные цифры (см. рис. 1.13a).

Что касается небольших цветных изображений, то наиболее известен набор данных **CIFAR** [KH09][1]. Он содержит 60 000 изображений хорошо знакомых объектов из 10 или 100 классов, примеры см. на рис. 1.13b.

1.5.1.2. ImageNet

Набольшие наборы данных хороши для апробирования идей, но важно также тестировать методы на более крупных наборах – с точки зрения как размера изображений, так и количества помеченных примеров. Самым употребительным из таких наборов является **ImageNet** [Rus+15]. Он содержит порядка 14 млн изображений размера 256×256×3, иллюстрирующих различные объекты из 20 000 классов, примеры см. на рис. 1.14a.

Набор данных ImageNet был положен в основу конкурса ImageNet Large Scale Visual Recognition Challenge (**ILSVRC**), проводившегося с 2010 по 2018 год. В нем использовалось подмножество, содержащее 1.3 млн изображений, принадлежащих 1000 классов. За эти годы сообщество добилось значительного прогресса, как показывает рис. 1.14b. В частности, в 2015 году CHC впервые превзошла человека (или, по крайней мере, одного человека, Андрея Карпатого) в задаче классификации изображений из набора ImageNet. Заметим, что это не означает, что CHC видят лучше человека (см., например, работу

[1] CIFAR означает Canadian Institute For Advanced Research. Эта организация финансировала разметку набора данных, за основу которого взят набор данных TinyImages по адресу http://groups.csail.mit.edu/vision/TinyImages/, созданный Антонио Торралба. Детали см. в [KH09].

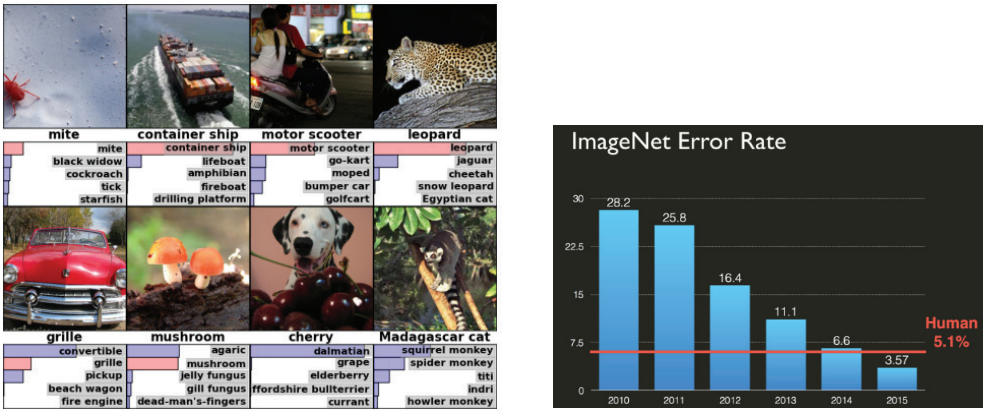
[YL21], где описаны некоторые типичные ошибки). Скорее всего, это отражает тот факт, что в наборе данных много нюансов **детальной классификации**, например различий между «тигром» и «тигровой кошкой», которые человеку просто непонятны. С другой стороны, достаточно гибкие CHC могут обучиться любым паттернам, в том числе случайным меткам [Zha+17a].

Рис. 1.13 ❖ (a) Визуализация набора данных Fashion-MNIST [XRV17]. Этот набор данных имеет такой же размер, как MNIST, но труднее для классификации. Всего имеется 10 классов: футболка/топ, брюки, пуловер, платье, куртка, сандалии, рубашка, кроссовки, сумка, полусапоги. Показаны первые 25 изображений из обучающего набора. Построено программой по адресу figures.probml.ai/book1/1.13. (b) Несколько изображений из набора данных CIFAR-10 [KH09]. Все изображения имеют размер $32 \times 32 \times 3$, где последнее измерение (3) означает формат RGB. Набор содержит 50k обучающих и 10k тестовых примеров. Имеется 10 классов: самолет, автомобиль, птица, кошка, собака, лягушка, лошадь, судно и грузовик. Показаны первые 25 изображений из обучающего набора. Построено программой по адресу figures.probml.ai/book1/1.13

Хотя набор ImageNet гораздо труднее MNIST и CIFAR в роли эталона классификации, он тоже дошел почти до точки «насыщения» [Bey+20]. Тем не менее качество работы того или иного метода на наборе ImageNet зачастую оказывается на удивление хорошим предиктором качества на других, никак не связанных задачах классификации изображений (см., например, [Rec+19]), поэтому он по-прежнему широко используется.

1.5.2. Некоторые широко известные наборы текстовых данных

Машинное обучение часто применяется к текстам для решения самых разных задач. Эта область называется **обработкой естественного языка** (natural language processing – **NLP**) (детали см., например, в [JM20]). Ниже даны краткие описания нескольких наборов текстовых данных, используемых в этой книге.



(a) (b)

Рис. 1.14 ❖ (a) Примеры изображений из набора данных ImageNet [Rus+15]. Это подмножество состоит из 1.3M цветных обучающих изображений размером 256×256 пикселей. Всего имеется 1000 меток, каждому изображению сопоставляется одна метка, а задача заключается в том, чтобы гарантировать попадание правильной метки в пять предсказаний с наибольшей вероятностью. Под каждым изображением показана истинная метка и распределение вероятностей первых пяти предсказанных меток. Если истинная метка оказывается среди первых пяти, то соответствующий ей столбик окрашен красным. Предсказания сгенерированы сверточной нейронной сетью (CHC) AlexNet (раздел 14.3.2). Заимствовано из работы [KSH12] (рис. 4). Печатается с разрешения Алекса Крижевски. (b) Изменение коэффициента неправильной классификации (непопадание в первые 5) в конкурсах ImageNet со временем. Печатается с разрешения Андрея Карпатого

1.5.2.1. Классификация текста

Простой задачей NLP является классификация текста, ее можно использовать для **классификации спама**, **анализа эмоциональной окраски** (например, является ли обзор фильма или товара положительным или отрицательным) и т. д. Для оценки таких методов часто используется **набор данных IMDB**, содержащий обзоры фильмов (см. [Maa+11]) (IMDB означает Internet Movie Database). В нем 25 000 помеченных примеров для обучения и 25 000 для тестирования. Каждый пример сопровождается бинарной меткой: положительная или отрицательная оценка. Примеры предложений приведены в табл. 1.3.

Таблица 1.3. Примеры двух первых предложений из набора данных IMDB, содержащего обзоры фильмов

1. this film was just brilliant casting location scenery story direction everyone's really suited the part they played robert <UNK> is an amazing actor ...
2. big hair big boobs bad music and a giant safety pin these are the words to best describe this terrible movie i love cheesy horror movies and i've seen hundreds...

Первый пример помечен как положительный, второй как отрицательный (<UNK> означает неизвестную лексему).

1.5.2.2. Машинный перевод

Более трудная задача NLP – обучиться отображать предложение ***x*** на одном языке в «семантически эквивалентное» предложение ***y*** на другом языке; это называется **машинным переводом**. Для обучения таких моделей нужны соответственные пары (***x***, ***y***). По счастью, существует несколько таких наборов данных, например от канадского парламента (англо-французские пары) и Европейского союза (Europarl). Подмножество последнего, **набор данных WMT** (Workshop on Machine Translation), состоит из англо-немецких пар и широко используется в качестве эталона для проверки алгоритмов.

1.5.2.3. Другие задачи типа seq2seq

Обобщение машинного перевода – задача об обучении отображения одной последовательности ***x*** в другую последовательность ***y***. Она называется **моделью seq2seq** и может рассматриваться как форма многомерной классификации (подробности см. в разделе 15.2.3). Эта постановка задачи очень общая, поэтому можно выделить много частных случаев: **реферирование документов**, **вопросно-ответные системы** и т. д. Например, в табл. 1.4 показано, как поставить задачу о вопросно-ответной системе как проблему seq2seq: входом является текст T и вопрос Q, а выходом – ответ A, представляющий собой множество слов, возможно, извлеченных из входного текста.

Таблица 1.4. Пары вопрос-ответ для примера текста из набора данных SQuAD

T: In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under **gravity**. The main forms of precipitation include drizzle, rain, sleet, snow, **graupel** and hail… Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals **within a cloud**. Short, intense periods of rain in scattered locations are called “showers”.
Q1: What causes precipitation to fall? A1: **gravity**
Q2: What is another main form of precipitation besides drizzle, rain, snow, sleet and hail? A2: **graupe**
Q3: Where do water droplets collide with ice crystals to form precipitation? A3: **within a cloud**

Каждый ответ представляет собой отрывок текста. Эту задачу можно решить с помощью разметки пар предложений. Входом является текстовый абзац T и вопрос Q. На выходе помечаются те слова T, которые отвечают на вопрос Q. Взято из работы [Raj+16]. Печатается с разрешения Перси Лианга.

1.5.2.4. Языковое моделирование

Под очень многоплановым термином «**языковое моделирование**» понимается задача создания безусловных порождающих моделей текстовых предложений, $p(x_1, \ldots, x_T)$. Для нее требуются только входные предложения ***x***, без соответственных «меток» ***y***. Поэтому проблему можно рассматривать как одну из форм обучения без учителя, обсуждавшегося в разделе 1.3. Если языковая модель порождает в ответ на вход какой-то выход, как в случае seq2seq, то ее можно рассматривать как условную порождающую модель.

1.5.3. Предобработка дискретных входных данных

Во многих моделях МО предполагается, что данные представляют собой вещественные векторы признаков, $\boldsymbol{x} \in \mathbb{R}^D$. Но иногда входные данные могут содержать дискретные признаки, например категориальные величины типа расы и пола или слова из некоторого словаря. Ниже мы обсудим некоторые способы предобработки таких данных с целью преобразования их в векторную форму. Эта операция часто применяется в различных моделях.

1.5.3.1. Унитарное кодирование

Категориальные признаки необходимо преобразовать в какую-то числовую шкалу, чтобы вычисление их взвешенных комбинаций имело смысл. Стандартный способ такого преобразования называется **унитарным кодированием**, или **индикаторным кодированием** (dummy encoding). Если переменная x может принимать K значений, то результатом ее унитарного кодирования будет one-hot$(x) = [\mathbb{I}(x = 1), \ldots, \mathbb{I}(x = K)]$. Например, если имеется 3 цвета (скажем, красный, зеленый и синий), то соответствующие им унитарные векторы имеют вид one-hot(красный) = [1, 0, 0], one-hot(зеленый) = [0, 1, 0], one-hot(синий) = [0, 0, 1].

1.5.3.2. Перекрестные произведения признаков

Линейная модель, в которой используется унитарное кодирование всех категориальных переменных, может уловить **основной эффект** каждой переменной, но не **эффекты взаимодействия** переменных. Например, предположим, что требуется предсказать топливную эффективность транспортного средства по двум категориальным переменным: типу (скажем, городской внедорожник, грузовик или семейный автомобиль) и стране происхождения (скажем, США или Япония). Если конкатенировать унитарные коды тернарных и бинарных признаков, то входные данные будут представлены в следующем виде:

$$\boldsymbol{\phi}(\boldsymbol{x}) = [1; \mathbb{I}(x_1 = S); \mathbb{I}(x_1 = T); \mathbb{I}(x_1 = F); \mathbb{I}(x_2 = U); \mathbb{I}(x_2 = J)], \tag{1.34}$$

где x_1 – тип, а x_2 – страна происхождения.

Эта модель не может уловить зависимости между признаками. Например, мы ожидаем от грузовиков меньшей топливной эффективности, но, быть может, американские грузовики даже менее эффективны, чем японские. Линейная модель (1.34) не может уловить этот факт, потому что вклад страны происхождения не зависит от типа автомобиля.

Это можно исправить, вычислив явные **перекрестные произведения признаков**. Например, мы можем определить новый составной признак с 3×2 возможными значениями, который будет улавливать взаимодействие между типом и страной происхождения. Тогда новая модель принимает вид:

$$f(\boldsymbol{x};\boldsymbol{w}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{\phi}(\boldsymbol{x}) \tag{1.35}$$

$$\begin{aligned}&= w_0 + w_1\mathbb{I}(x_1 = S) + w_2\mathbb{I}(x_1 = T) + w_3\mathbb{I}(x_1 = F)\\ &+ w_4\mathbb{I}(x_2 = U) + w_5\mathbb{I}(x_2 = J)\\ &+ w_6\mathbb{I}(x_1 = S; x_2 = U) + w_7\mathbb{I}(x_1 = T; x_2 = U) + w_8\mathbb{I}(x_1 = F; x_2 = U)\\ &+ w_9\mathbb{I}(x_1 = S; x_2 = J) + w_{10}\mathbb{I}(x_1 = T; x_2 = J) + w_{11}\mathbb{I}(x_1 = F; x_2 = J).\end{aligned} \tag{1.36}$$

Видно, что перекрестные произведения позволяют преобразовать исходный набор данных в **широкий формат** с гораздо большим числом столбцов.

1.5.4. Предобработка текстовых данных

В разделе 1.5.2 мы кратко обсудили классификацию текста и другие задачи NLP. Для подачи текста на вход классификатору нужно решить ряд вопросов. Во-первых, документы имеют переменную длину, а не являются векторами признаков фиксированной длины, как предполагается во многих моделях. Во-вторых, слова – это категориальные переменные, имеющие много значений (столько, сколько слов в словаре), поэтому размерность соответствующей унитарной кодировки будет очень велика, а в многомерном пространстве нет естественного понятия сходства. В-третьих, во время тестирования могут встретиться слова, которые не предъявлялись на этапе обучения (так называемые **внесловарные слова**, или OOV-слова). Некоторые решения этих проблем мы обсудим ниже, а дополнительные сведения можно найти, например, в работах [BKL10; MRS08; JM20].

1.5.4.1. Модель мешка слов

Простой подход к обработке текстовых документов переменной длины состоит в том, чтобы интерпретировать их как **мешок слов**, в котором порядок слов игнорируется. Чтобы преобразовать такой документ в вектор из фиксированного пространства входов, мы сначала преобразуем каждое слово в **лексему** из некоторого словаря.

Для уменьшения количества лексем применяются различные методы предобработки, например: отбрасывание знаков препинания, преобразование всех слов в нижний регистр, отбрасывание часто встречающихся, но не информативных слов типа «and» и «the» (это называется **исключением стоп-слов**), замена слов основной формой, например замена «running» и «runs» формой «run» (это называется **стеммингом**) и т. д. Детали см., например, в [BL12], а демонстрационный код на странице code.probml.ai/book1/text_preproc_torch.

Обозначим x_{nt} лексему в позиции t n-го документа. Если в словаре имеется D различных лексем, то n-й документ можно представить D-мерным вектором $\tilde{\boldsymbol{x}}_n$, где $\tilde{x}_{nv}$ – количество вхождений слова v в документ n:

$$\tilde{x}_{nv} = \sum_{t=1}^{T} \mathbb{I}(x_{nt} = v), \tag{1.37}$$

где T – длина документа n. Теперь документы можно интерпретировать как векторы в пространстве $\mathbb{R}^D$. Это так называемая **модель векторного пространства** [SWY75; TP10].

Традиционно входные данные хранятся в матрице плана **X** размера $N \times D$, где D – число признаков. Но в контексте моделей векторного пространства принято представлять входные данные **терм-документной матрицей** размера $D \times N$, где TF_{ij} – частота терма i в документе j (см. рис. 1.15).

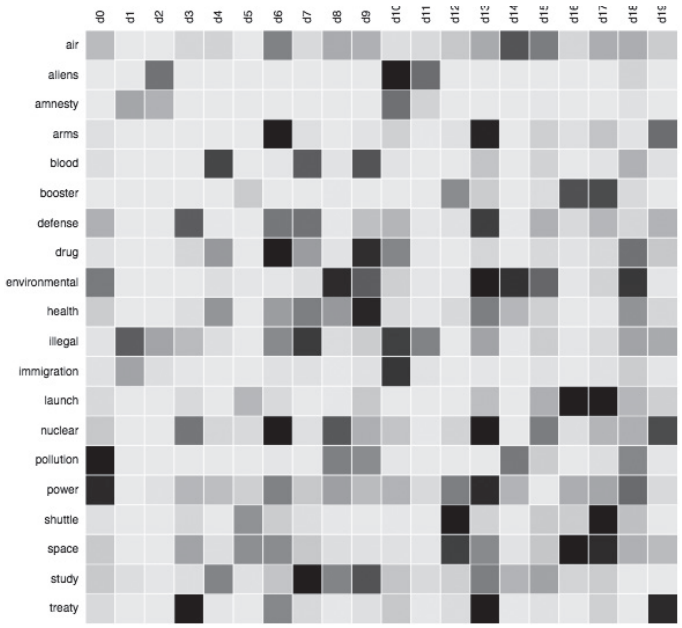


Рис. 1.15 ❖ Пример терм-документной матрицы, в которой исходные счетчики заменены значениями TF-IDF (см. раздел 1.5.4.2). Чем темнее ячейка, тем больше значение. Взято из статьи https://bit.ly/2kByLQI. Печатается с разрешения Кристофа Карла Клинга

1.5.4.2 TF-IDF

Одной из проблем, сопутствующих представлению документов векторами счетчиков слов, является тот факт, что часто встречающиеся слова могут оказывать несоразмерное их важности влияние, просто потому что счетчик вхождений для них больше, хотя никакого особого семантического смысла они не несут. Стандартное решение этой проблемы – заменить счетчики логарифмами, что уменьшает влияние слов, многократно встречающихся в одном документе.

А чтобы уменьшить влияние слов, которые вообще часто встречаются (во всех документах), мы вычисляем так называемую **обратную частоту документа**, которая определяется следующим образом: $\mathrm{IDF}_i \triangleq \log N/(1 + \mathrm{DF}_i)$, где DF_i – число документов, содержащих терм i. Оба преобразования можно объединить и вычислить матрицу **TF-IDF**:

$$\text{TFIDF}_{ij} = \log(\text{TF}_{ij} + 1) \times \text{IDF}_i \tag{1.38}$$

(дополнительно строки часто нормируются). Такая матрица дает семантически более осмысленное представление документов и используется в качестве входа во многих алгоритмах МО. Пример см. по адресу code.probml.ai/book1/tfidf_demo.

1.5.4.3. Погружения слов

Хотя преобразование в матрицу TF-IDF улучшает векторное представление слов, придавая больший вес «информативным» словам и меньший «неинформативным», оно не решает фундаментальную проблему, заключающуюся в том, что семантически родственные слова, например «man» и «woman», могут располагаться в векторном пространстве дальше друг от друга, чем семантически не связанные слова, например «man» и «banana». Поэтому предположение о том, что для точек, близких в пространстве входов, и выходы будут близки, неявно подразумеваемое в моделях логистической регрессии, неверно.

Стандартный способ решения этой проблемы – использование **погружений слов** (word embeddings), когда каждый разреженный унитарный вектор $\boldsymbol{x}_{nt} \in \{0, 1\}^V$ отображается в плотный вектор меньшей размерности, $\boldsymbol{e}_{nt} \in \mathbb{R}^K$ посредством преобразования $\boldsymbol{e}_{nt} = \mathbf{E}\boldsymbol{x}_{nt}$, где матрица $\mathbf{E}$ обучена таким образом, что семантически родственные слова оказываются рядом. В разделе 20.5 мы увидим, что существует много способов обучиться такому погружению.

Имея матрицу погружений, мы можем представить текстовый документ переменной длины в виде **мешка погружений слов**. Затем этот мешок преобразуется в вектор фиксированной длины путем суммирования (или усреднения) погружений:

$$\bar{\boldsymbol{e}}_n = \sum_{t=1}^{T} \boldsymbol{e}_{nt} = \mathbf{E}\tilde{\boldsymbol{x}}_n, \tag{1.39}$$

где $\tilde{\boldsymbol{x}}_n$ – представление в виде мешка слов из формулы (1.37). Далее этот вектор можно использовать в классификаторе методом логистической регрессии, который мы кратко описали в разделе 1.2.1.5. Итоговая модель принимает вид:

$$p(y = c|\boldsymbol{x}_n; \boldsymbol{\theta}) = \mathcal{S}_c(\mathbf{WE}\tilde{\boldsymbol{x}}_n). \tag{1.40}$$

Мы часто используем **предварительно обученную матрицу погружений E**, и в этом случае модель линейна относительно $\mathbf{W}$, что упрощает оценивание параметров (см. главу 10). Смотрите также обсуждение контекстуального погружения слов в разделе 15.7.

1.5.4.4. Обработка новых слов

Во время тестирования модели могут встретиться совершенно новые слова, которые раньше не предъявлялись. Такие слова называются **внесловарны-**

ми (out of vocabulary – **OOV**). Их появление неизбежно, потому что множество слов – **открытый класс**. Так, множество имен собственных (имен людей и названий мест) не ограничено.

Стандартная эвристика для решения этой проблемы – заменить все новые слова специальным символов **UNK** (unknown – неизвестно). Но при этом теряется информация. Например, встретив слово «athazagoraphobia», мы можем догадаться, что это «боязнь чего-то», поскольку производный от греческого слова суффикс «phobia» часто встречается в английском языке и означает «боязнь» (кстати говоря, «атазагорафобия» означает боязнь быть забытым или проигнорированным).

Мы могли бы работать на уровне символов, но тогда пришлось бы обучать модель, как группировать частые комбинации букв в слова. Лучше воспользоваться тем фактом, что у слов есть внутренняя структура, и брать в качестве входных данных **подслова**, или **части слов** [SHB16; Wu+16]; они часто создаются методом **кодирования пар байтов** [Gag94], который представляет собой вариант сжатия данных, когда для представления общих подстрок создаются новые символы.

1.5.5. Обработка отсутствующих данных

Иногда некоторые данные отсутствуют, т. е. части входа $\boldsymbol{x}$ или выхода $\boldsymbol{y}$ неизвестны. Если во время обучения выход неизвестен, то пример останется не помеченным; такие сценарии обучения с частичным привлечением учителя рассматриваются в разделе 19.3. Поэтому мы сосредоточимся на случае, когда отсутствуют некоторые входные признаки, – на этапе обучения, этапе тестирования или на обоих.

Для моделирования этой ситуации обозначим $\mathbf{M}$ матрицу бинарных переменных размера $N \times D$, в которой $M_{nd} = 1$, если признак d в примере n отсутствует, и $M_{nd} = 0$ в противном случае. Пусть $\mathbf{X}_v$ – присутствующие части матрицы входных признаков, соответствующие элементам $M_{nd} = 0$, а $\mathbf{X}_h$ – отсутствующие части, соответствующие элементам $M_{nd} = 1$. Обозначим $\mathbf{Y}$ матрицу выходных меток и предположим, что она наблюдаема целиком. Если $p(\mathbf{M}|\mathbf{X}_v, \mathrm{X}_h, \mathbf{Y}) = p(\mathbf{M})$, то говорят, что данные **отсутствуют вполне случайно** (missing completely at random – **MCAR**), поскольку присутствие или отсутствие данных вообще не зависит от скрытых или наблюдаемых признаков. Если $p(\mathbf{M}|\mathbf{X}_v, \mathrm{X}_h, \mathbf{Y}) = p(\mathbf{M}|\mathbf{X}_v, \mathbf{Y})$, то говорят, что данные **отсутствуют случайно** (missing at random – **MAR**), поскольку отсутствие не зависит от скрытых признаков, но может зависеть от наблюдаемых. Если ни то, ни другое условие не выполняется, то говорят, что данные **отсутствуют не случайно** (not missing at random – **NMAR**).

В случаях MCAR и MAR мы можем игнорировать механизм отсутствия, поскольку он ничего не говорит о скрытых признаках. Но в случае NMAR необходимо смоделировать **механизм отсутствия данных**, потому что само отсутствие информации может нести полезную информацию. Например, тот факт, что человек не дал ответа на деликатный вопрос в анкете (например, «Болеете ли вы ковидом?»), может дать информацию об истинном значе-

нии. Дополнительные сведения о моделях отсутствия данных см. в работах [LR87; Mar08].

В этой книге мы всегда будем придерживаться предположения MAR. Но даже в этом предположении мы не можем напрямую воспользоваться дискриминантной моделью, например ГНС, при наличии отсутствующих входных признаков, потому что вход $\boldsymbol{x}$ будет иметь неизвестные значения. Стандартная эвристика, применяемая в этом случае, называется **подстановкой среднего значения** – вместо отсутствующих значений включаются их эмпирические средние. Вообще, мы можем аппроксимировать входные данные порождающей моделью и использовать эту модель для **восполнения** отсутствующих значений. Подходящие для этой цели порождающие модели мы кратко обсудим в главе 20, а более полно во втором томе этой книги, [Mur22].

1.6. Обсуждение

В этом разделе мы определим место МО и этой книги в более общем контексте.

1.6.1. Связь МО с другими дисциплинами

Есть несколько сообществ, работающих в связанных с МО областях, называемых по-разному. **Прогнозная аналитика** напоминает обучение с учителем (в особенности классификацию и регрессию), но в большей степени относится к бизнес-приложениям. **Добыча данных** охватывает машинное обучение с учителем и без учителя, но применяется в основном к структурированным данным, которые обычно хранятся в крупных коммерческих базах данных. В **науке о данных** (data science) применяются методы машинного обучения и статистики, но много внимания уделяется также другим вопросам, в частности интеграции данных, визуализации данных и работе со специалистами в предметной области, зачастую в итеративном цикле с обратной связью (см., например, [BS17]). Различие между этими дисциплинами преимущественно терминологическое[1].

МО также тесно связано с математической **статистикой**. Действительно, Джерри Фридман, известный профессор статистики в Стенфордском университете, писал[2]:

> [Если бы статистика] с самого начала включала методологию вычислений в качестве основополагающего инструмента, а не просто была удобным способом приложения имеющихся у нас инструментов, то нужды во многих других дисциплинах, связанных с данными [например, МО], и не возникло бы – они просто были бы частью статистики.
>
> — *Джерри Фридман* [Fri97b]

[1] См. полезный «Глоссарий МО» на страницу https://developers.google.com/machine-learning/glossary/.

[2] Цитируется по https://brenocon.com/blog/2008/12/statistics-vs-machine-learning-fight/.

Машинное обучение также связано с **искусственным интеллектом** (**ИИ**). Исторически предполагалось, что мы сможем программировать «интеллект» вручную (см., например, [RN10; PM17]), но этот подход не выдержал столкновения с реальностью, в основном потому что явно закодировать все знания, необходимые таким системам, оказалось слишком трудно. Поэтому возродился интерес к МО как к средству помочь ИИ в приобретении собственных знаний. (На самом деле связи между МО и ИИ настолько тесные, что зачастую эти термины употребляют как синонимы, хотя это, пожалуй, только вводит в заблуждение [Pre21].)

1.6.2. Структура книги

Мы видели, что МО тесно связано со многими другими разделами математики, статистики, информатики и т. д. Поэтому трудно решить, с чего начать.

В этой книге мы выбрали один конкретный путь по этому ландшафту взаимосвязанных дисциплин, используя теорию вероятностей в качестве объединяющей призмы. В части I мы излагаем основы статистики, в частях II–IV рассматриваем обучение с учителем, а в части V – обучение без учителя. Дополнительные сведения об этих (и других) вопросах читатель найдет во втором томе этой книги, [Mur22],

В дополнение к книге на сайте http://probml.ai имеются Python-блокноты.

1.6.3. Подводные камни

В этой книге мы увидим, как можно использовать машинное обучение для создания систем, которые могут (или хотя бы пытаются) предсказывать выходы по заданным входам, а затем использовать предсказания для выбора действий с целью минимизировать ожидаемую потерю. При разработке таких систем бывает трудно спроектировать функцию потерь, которые точно отражает все наши предпочтения; это может приводить к «**манипулированию вознаграждением**» (reward hacking) – машина усердно оптимизирует заданную нами функцию вознаграждения, а потом обнаруживается, что функция не улавливает различных ограничений или предпочтений, которые мы забыли описать [Wei76; Amo+16; D'A+20]. (Это особенно важно, когда необходимо искать компромиссы между несколькими целями.)

Манипулирование вознаграждением поднимает различные вопросы в контексте **этики** и **безопасности ИИ** (см., например, [KR19; Lia20]). В работе [Rus19] предлагается решать эту проблему, отказавшись от задания функции вознаграждения, а заставив вместо этого машину выводить вознаграждение путем наблюдения за поведением человека; этот подход называется **обратным обучением с подкреплением**. Однако слишком точное подражание текущему или прошлому поведению человека может быть нежелательно, на него могут повлиять доступные для обучения данные (см., например, [Pau+20]).

Другой подход – рассматривать МО как инструмент для построения адаптивных компонентов, являющихся частями более крупной системы. Такую систему следует проектировать и регулировать так же, как мы проектируем и регулируем другие сложные полуавтономные продукты человеческой деятельности: самолеты, платформы для интернет-трейдинга или системы медицинской диагностики (см. [Jor19]). МО играет важную роль в таких системах, но необходимы дополнительные сдержки и противовесы, чтобы закодировать наши предшествующие знания и предпочтения и гарантировать, что система будет действовать так, как запланировано.