1 Алгоритмы машинного обучения

В этой главе

- ✓ Типы алгоритмов машинного обучения
- ✓ Важность изучения алгоритмов с нуля
- ✓ Введение в байесовский вывод и глубокое обучение
- ✓ Программная реализация алгоритмов машинного обучения с нуля

Алгоритм — это последовательность шагов, необходимых для выполнения определенной задачи. Он получает входные данные, выполняет последовательность операций и выдает желаемый результат. Простейшим примером алгоритма является *сортировка* списка целых чисел. После выполнения набора операций мы получаем отсортированный список. Информация в таком списке приобретает более упорядоченный вид, и в ней становится легче находить ответы на интересующие нас вопросы.

Обычно алгоритм, в зависимости от размера входных данных n, характеризуется двумя показателями: тем, как быстро он работает (временна́я сложность алгоритма), и тем, сколько памяти ему требуется (пространственная сложность алгоритма). Например, сортировка на основе сравнения, как мы увидим позже, обладает временно́й сложностью $O(n \log n)$ и требует памяти $O(n)$.

Существует множество способов провести сортировку, но в каждом случае автор алгоритма, если он придерживается классической парадигмы, перечисляет набор инструкций. Представьте себе ситуацию, в которой вы можете *выучить*

эти инструкции, используя ряд доступных вам примеров входных и выходных данных. Именно так выглядит структура алгоритмической парадигмы машинного обучения. Это подобно тому, как учится наш мозг, когда мы играем в игру «соедини по точкам» или рисуем пейзаж: мы заполняем пропуски, сравнивая на каждом шаге желаемое и имеющееся. Так в общих чертах работают алгоритмы ML с учителем (supervised). В процессе обучения алгоритмы ML выучивают правила (например, границы классификации) на основе обучающих примеров путем оптимизации целевой функции (objective function). Во время тестирования алгоритмы ML применяют ранее выученные правила к новым входным данным и выдают прогноз, как показано на рис. 1.1.

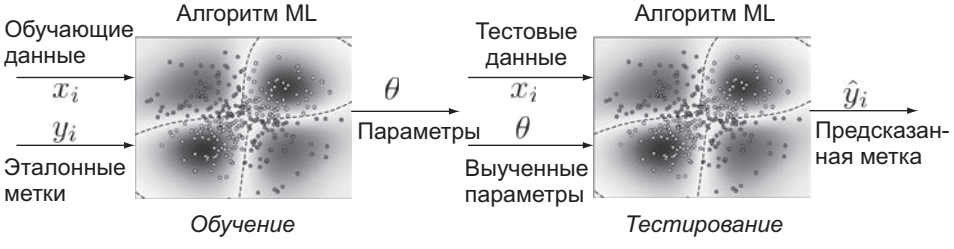

Рис. 1.1. Обучение с учителем: обучение (слева) и тестирование (справа)

1.1. Типы алгоритмов машинного обучения

Давайте немного раскроем смысл предыдущего абзаца и введем некоторые обозначения. Эта книга посвящена алгоритмам ML, которые можно сгруппировать по следующим категориям: обучение с учителем (supervised), обучение без учителя (unsupervised) и глубокое обучение (deep learning). При *обучении с учителем* задача состоит в том, чтобы получить отображение f входных данных x на выходные y при заданном обучающем наборе данных $D = \{(x_1, y_1), ..., (x_n, y_n)\}$, состоящем из n пар вход-выход. Другими словами, нам дается n примеров того, как должны выглядеть выходные данные при заданных входных. Выходное значение y также часто называют *меткой* (label). Оно представляет собой контрольный сигнал, который сообщает нашему алгоритму, каков правильный ответ.

Обучение с учителем, в зависимости от той величины, которую мы пытаемся предсказать, в свою очередь, можно разделить на *классификацию* и *регрессию*. Если наше предсказание y является дискретной величиной (например, K различных классов), то перед нами стоит задача классификации. В то же время если наше выходное значение y — непрерывная величина (к примеру, вещественное число вроде курса акций), то мы сталкиваемся с проблемой регрессии.

Таким образом, характер задачи меняется в зависимости от величины y, которую мы пытаемся предсказать. В любом случае мы хотим максимально приблизиться к эталонному (ground truth) значению y.

Обычно для оценки эффективности или близости к эталону используется *функция потерь* (loss function). Она позволяет рассчитать расстояние между предсказанной и истинной метками. Пусть $y = f(x; \theta)$ — наш алгоритм ML, который отображает входные примеры x на выходные метки y, параметризуемый θ, где θ содержит все требуемые параметры нашего алгоритма.

Теперь мы можем задать функцию потерь для задачи классификации, равную количеству неправильно классифицированных случаев, как показано в формуле 1.1:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n} 1\left[y_i \neq f(x_i; \theta)\right]. \tag{1.1}$$

Здесь 1[] — это индикаторная функция, которая принимает значение 1, когда аргумент внутри скобок истинен, и 0 в противном случае. Выражение в формуле 1.1 означает, что мы суммируем все случаи, в которых наше предсказание $f(x_i; \theta)$ не соответствовало эталонной метке y_i, и делим сумму на общее количество примеров n. Иначе говоря, мы вычисляем среднюю долю случаев ошибочной классификации. Наша цель — минимизировать функцию потерь (то есть найти такой набор параметров θ, при котором коэффициент ошибочной классификации будет как можно ближе к нулю). Обратите внимание, что для задачи классификации существует множество альтернативных функций потерь, таких как кросс-энтропия, которые мы рассмотрим в последующих главах.

Для непрерывных меток, или выходных переменных, обычно в качестве функции потерь используют *среднеквадратичное отклонение* (mean squared error, MSE). Оно определяет, насколько наша оценка далека от эталона, как показано в формуле 1.2:

$$L(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left[y_i - f(x_i; \theta)\right]^2. \tag{1.2}$$

Как видно из формулы, мы вычитаем наше предсказание из истинной метки, возводим разность в квадрат и усредняем результат по всем значениям. Возводя в квадрат, мы избавляемся от отрицательных значений и выбраковываем случаи с большим отклонением от эталона.

Одной из главных задач ML является умение делать обобщение примеров, которые не встречались до этого. Мы хотим добиться высокой доли верных результатов (accuracy), то есть низких потерь, не только для обучающих данных (уже размеченных), но и для новых незнакомых тестовых примеров. Именно эта способность к обобщению делает ML таким привлекательным: если мы сможем разработать алгоритмы ML, которые смогут выходить за рамки своей обучающей программы, то мы станем на шаг ближе к общему (близкому к человеческому) искусственному интеллекту (artificial general intelligence, AGI).

При *обучении без учителя* не только отсутствуют метки y, но перед нами даже не стоит задача нахождения соответствия между входными и выходными примерами;

вместо этого мы хотим разобраться в самих данных, то есть обнаружить в них закономерности, а это проще сделать, если спроецировать многомерные данные в пространство меньшей размерности, как показано на рис. 1.2. Таким образом, в случае обучения без учителя набор обучающих данных состоит из n входных примеров без каких-либо меток y: $D = \{x_1,...,x_n\}$. Простейший пример обучения без учителя — это поиск кластеров в данных. Интуиция нам подсказывает, что точки данных, которые принадлежат к одному и тому же кластеру, имеют сходные характеристики. И действительно, центр кластера может выступать как типичный представитель точек этого кластера, заменяя их, что используется в алгоритмах сжатия. С другой стороны, анализ расстояния между кластерами в пространстве меньшей размерности может помочь нам оценить взаимосвязь между различными группами. Кроме того, точка, удаленная от всех существующих кластеров, может рассматриваться как аномалия, что подводит к алгоритму обнаружения аномалий. Как можно заметить, обучение без учителя имеет бесконечное число интересных вариантов использования, и на всем протяжении этой книги самые захватывающие алгоритмы из этой серии мы будем разбирать с нуля.

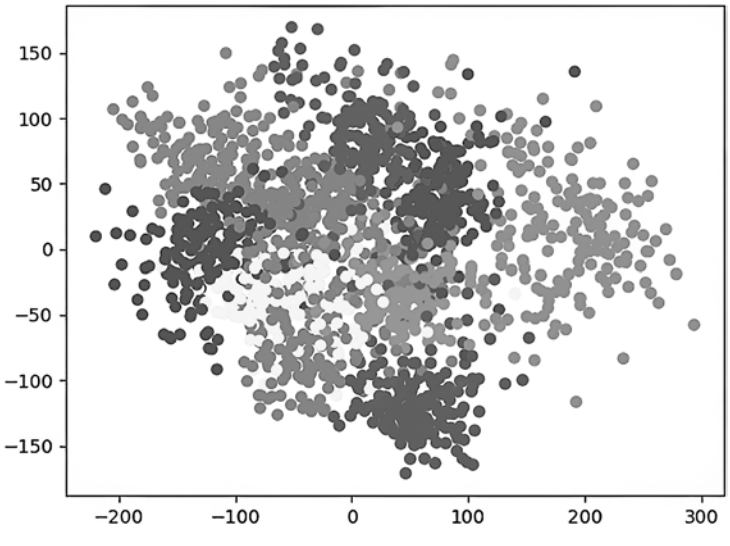

Рис. 1.2. Обучение без учителя: кластеры точек данных, спроецированные в двумерное пространство

Еще одной очень важной областью применения современных алгоритмов ML является *глубокое обучение*. Название происходит от стека вычислительных уровней, вместе образующих вычислительный граф. Глубина этого графа относится к последовательным вычислениям, а ширина — к параллельным.

Как мы увидим в дальнейшем, модели глубокого обучения постепенно уточняют свои параметры с помощью алгоритма обратного распространения ошибки

(back-propagation) до тех пор, пока не достигнут целевой функции. Такие модели получили широкое распространение в отрасли благодаря своей способности решать сложные задачи с высокой долей верных результатов. В качестве примера на рис. 1.3 изображена архитектура глубокого обучения для анализа тональности текста (sentiment analysis). В следующих главах мы узнаем больше о том, что представляет собой каждый из ее блоков.

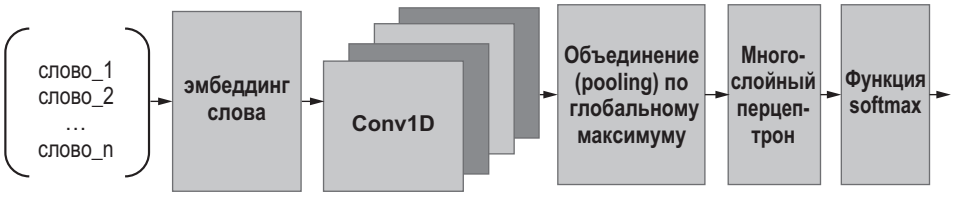

Рис. 1.3. Архитектура глубокой нейронной сети (DNN) для анализа тональности текста

Глубокое обучение — стремительно развивающаяся область исследований, которой в данной книге уделяется особое внимание. Например, при обучении с самоконтролем (self-supervised learning), используемым в трансформерных моделях, мы задействуем контекст и структуру естественного языка в качестве контрольного сигнала, тем самым извлекая метки из самих данных. В дополнение к классическим сферам применения глубокого обучения, таким как обработка естественного языка (natural language processing, NLP) и компьютерное зрение (computer vision, CV), мы также обсудим генеративные модели, научимся прогнозировать на основе данных временны́х рядов и работать с реляционными графами.

1.2. Зачем изучать алгоритмы с нуля?

Глубокое понимание алгоритмов ML даст читателю несколько ценных преимуществ. Во-первых, вы сможете подбирать правильный подход для решения поставленных задач. Зная внутреннее устройство алгоритма, вы сумеете понять его недостатки, допущения, сделанные при его создании, а также преимущества в различных сценариях обработки данных. Такой навык позволит вам взвешенно подходить к выбору правильного решения задачи и экономить время, исключая заведомо неподходящие методы.

Во-вторых, сможете объяснить результаты работы данного алгоритма заинтересованным сторонам. Умение разъяснять и представлять результаты аудитории в производственных или академических кругах — важный навык специалиста по ML.

В-третьих, будете использовать интуицию, развитую при чтении этой книги, для решения сложных задач ML. Для того чтобы разбить подобную задачу на более

мелкие части и понять, где таится корень проблемы, часто требуется глубокое понимание фундаментальных основ и алгоритмическая проницательность. Эта книга научит читателя создавать работающие образцы с минимальным набором функций, а также на основе существующих алгоритмов разрабатывать и уметь отлаживать более сложные модели.

В-четвертых, сможете совершенствовать алгоритмы, чтобы они соответствовали изменяющимся условиям, в частности, в ситуации, когда формулы из учебника или библиотеки не могут быть применены в исходном виде. Глубокое понимание темы, которое вы разовьете, читая эту книгу, поможет модифицировать существующие алгоритмы в соответствии с вашими потребностями.

И наконец, часто возникает необходимость повысить производительность имеющихся моделей. Концепции, рассматриваемые в книге, позволят читателю сделать это. Таким образом, освоение алгоритмов ML с нуля поможет вам выбирать правильный вариант для решения поставленной задачи, объяснять получаемые результаты, решать запутанные проблемы, развивать методологию и повышать производительность существующих моделей.

1.3. Математические основы

Прежде чем приступать к основательному изучению алгоритмов ML, будет полезным повторить основные положения прикладной теории вероятностей, математического анализа и линейной алгебры. Хороший обзор теории вероятностей содержится в книге Димитриса Берцекаса (Dimitri Bertsekas) и Джона Цициклиса (John Tsitsiklis) «Introduction to Probability» (Athena Scientific, 2002). Читатель должен быть знаком с непрерывными и дискретными случайными величинами, условными и частными распределениями, правилом Байеса, цепями Маркова и предельными теоремами.

Для повторения положений математического анализа я рекомендую книгу Джеймса Стюарта (James Stewart) «Calculus» (Thomson Brooks / Cole, 2007). Предполагается, что читатель владеет правилами дифференцирования и интегрирования, имеет представление о последовательностях и рядах, векторах и стереометрии, частных производных и многомерных интегралах.

И наконец, книга Гилберта Стрэнга (Gilbert Strang) «Introduction to Linear Algebra» (Wellesley-Cambridge Press, 2016) служит отличным введением в линейную алгебру. От читателя требуется знакомство с векторными пространствами, матрицами, собственными значениями и собственными векторами, матричными нормами и факторизациями, положительно определенными и полуопределенными матрицами, а также матричным исчислением. В дополнение к вышеупомянутым книгам, пожалуйста, ознакомьтесь с приложением А, где содержится список рекомендуемых источников, способных углубить ваше понимание алгоритмов, представленных в этой книге.

1.4. Байесовский вывод и глубокое обучение

Байесовский вывод (Bayesian inference) дает возможность обновить наши представления о мире на основе наблюдений. В нашем сознании существует множество ментальных моделей, объясняющих различные аспекты окружающей действительности, и принимая во внимание новые данные, мы можем скорректировать наши неявные представления и тем самым улучшить понимание реальности. Любая *вероятностная модель* описывается набором управляющих ее поведением параметров θ, понимаемых как случайные величины, а также связанным набором данных x.

Цель байесовского вывода состоит в том, чтобы найти апостериорное распределение $p(\theta|x)$ (распределение по параметрам при заданных данных), которое хорошо отражает конкретный аспект реальности. Апостериорное распределение пропорционально произведению правдоподобия $p(x|\theta)$ (распределение по данным при фиксированных параметрах) и априорной вероятности $p(\theta)$ (начальное распределение по параметрам), что следует из правила Байеса, представленного в формуле 1.3:

$$\underbrace{p(\theta|x)}_{\text{Апостериорная вероятность}} = \frac{\overbrace{p(\theta|x)}^{\text{Правдоподобие}}\overbrace{p(\theta)}^{\text{Априорная вероятность}}}{\underbrace{p(x)}_{\text{Достоверность}}} = \frac{p(\theta|x)p(\theta)}{\underbrace{\int p(\theta|x)p(\theta)d\theta}_{\text{Функция разбиения Z}}} \sim p(x|\theta)p(\theta). \quad (1.3)$$

Априорное распределение $p(\theta)$ отражает наши исходные представления и может быть либо неинформативным (например, равномерным по всем возможным состояниям), либо информативным (например, основанным на опыте в определенной области). Более того, результаты нашего вывода зависят от выбранного априорного распределения: не только от значения параметров, но и от вида функциональной зависимости. Можно представить себе цепочку обновления наших представлений в виде байесовского механизма, в котором априорное распределение становится апостериорным по мере получения большего количества данных, как показано на рис. 1.4. Видно, как при поступлении новых данных априорное распределение превращается в апостериорное по правилу Байеса.

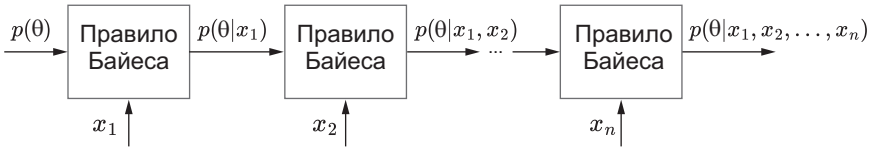

Рис. 1.4. Байесовский механизм, показывающий преобразование априорного распределения в апостериорное при получении новых данных

Априорные распределения, имеющие тот же вид, что и апостериорные, известны как *сопряженные априорные распределения*. Они, как правило, являются предпочтительными, поскольку упрощают вычисления благодаря выражениям в закрытой форме. Знаменатель $Z = p(x) = \int p(x|\theta)p(\theta)d\theta$ известен как *нормирующая постоянная*, или *функция распределения*, и он часто с трудом поддается вычислениям из-за интегрирования в пространстве параметров большой размерности. В этой книге мы рассмотрим несколько методов оценки Z.

Мы можем представить взаимосвязи между различными случайными величинами в нашей модели в виде графа, как показано на рис. 1.5, результатом чего является *вероятностная графовая модель* (ВГМ). Каждый узел графа представляет собой случайную величину (СВ), а каждое ребро — условную зависимость. Параметры модели изображены в виде прозрачных узлов, в то время как заштрихованные узлы представляют наблюдаемые данные, а прямоугольники обозначают копию или повторение случайной переменной. Сама топология графа меняется в зависимости от целей, которые вы преследуете при моделировании. Однако задачи байесовского вывода остаются прежними: найти апостериорное распределение по параметрам модели при фиксированных наблюдаемых данных.

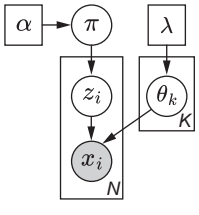

Рис. 1.5. Вероятностная графовая модель (ВГМ) для модели смеси нормальных распределений

В отличие от ВГМ, где связи задаются экспертами в предметной области, *модели глубокого обучения* автоматически изучают представления о мире с помощью алгоритма обратного распространения, который минимизирует целевую функцию. *Глубокие нейронные сети* (ГНС) (DNN, deep neural network) состоят из набора слоев, параметризованных матрицами весов и параметрами смещения. Математически DNN можно выразить как композицию функций отдельных слоев, как показано в формуле 1.4:

$$DNN = (x;\ \theta) = f_L\left(f_{L-1}\left(\cdots\left(f_1(x;\ \theta_1)\right)\cdots\right);\ \theta_L\right). \tag{1.4}$$

Здесь $f_1(x) = f(x; \theta_1)$ — функция в слое 1. Композиционная форма ГНС напоминает нам о цепном правиле для дифференцирования параметров в рамках стохастического градиентного спуска. На страницах этой книги мы будем рассматривать разные типы нейронных сетей: сверточные (convolutional neural networks, CNN), рекуррентные (recurrent neural networks, RNN), а также трансформеры и нейронные сети графовой структуры (graph neural networks, GNN), которые используются в самых различных областях, от компьютерного зрения до финансов.

1.4.1. Два основных направления байесовского вывода: MCMC и VI

Марковская цепь Монте-Карло (Markov chain Monte Carlo, MCMC) — это методология семплирования (sampling) в высокоразмерных пространствах параметров для аппроксимации апостериорного распределения $p(\theta|x)$. В статистике процесс *семплирования* означает генерацию случайной выборки значений из заданного распределения вероятностей. Существует множество подходов к семплированию в пространствах параметров большой размерности. Как мы увидим в последующих главах, в основе метода MCMC лежит построение цепи Маркова, стационарное распределение которой представляет собой интересующую нас целевую плотность вероятности (то есть апостериорное распределение). При случайном блуждании в пространстве состояний доля времени, которую мы проводим в каждом состоянии θ, будет пропорциональна $p(\theta|x)$. Как следствие, мы можем использовать интегрирование по методу Монте-Карло для получения интересующих величин, связанных с нашим апостериорным распределением.

Прежде чем мы обсудим высокоразмерные пространства параметров, давайте рассмотрим способы осуществлять выборку в случае низкоразмерных пространств. Наиболее популярный метод семплирования из одномерных распределений известен как *метод обратного преобразования* (inverse cumulative density function (CDF) method). Функция распределения определяется как $\mathrm{CDF}_X(x) = P(X \le x)$ (рис. 1.6).

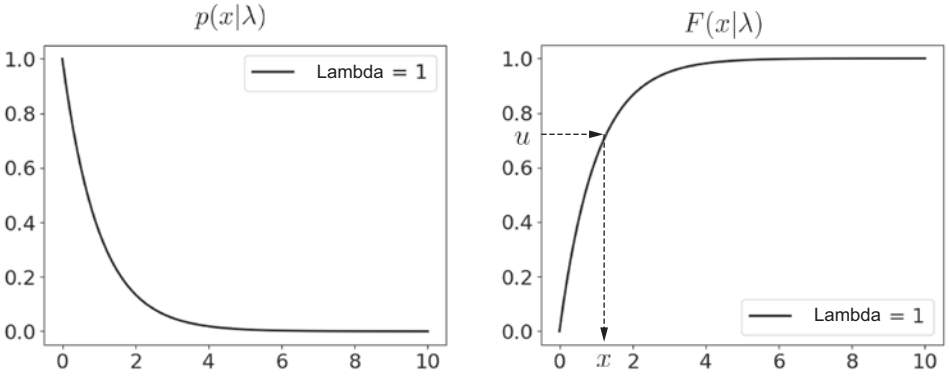

Рис. 1.6. Функция плотности вероятности (слева) и функция распределения (справа) экспоненциальной случайной величины (СВ)

Давайте для примера рассмотрим случай экспоненциальной случайной величины (СВ) с функцией плотности вероятности и функцией распределения, заданными формулами 1.5.

$$p(x\,|\,\lambda) = \lambda e^{-\lambda x},\ x \ge 0 \qquad F(x\,|\,\lambda) = \int_0^x p(x\,|\,\lambda)dx = 1 - e^{-\lambda x},\ x \ge 0. \qquad (1.5)$$

Обратная функция распределения описывается формулой 1.6:

$$F^{-1}(u) = -\frac{\ln(1-u)}{\lambda}. \tag{1.6}$$

Таким образом, чтобы сгенерировать выборку для экспоненциальной СВ, нам сначала нужно сгенерировать выборку из равномерного распределения $u \sim \text{Unif}(0, 1)$ и применить преобразование $-\ln(1 - u)/\lambda$. Генерируя достаточное количество семплов, мы можем достичь необходимой доли верных результатов. Установление того, как *эффективно* генерировать выборки из высокоразмерных распределений, — одна из проблем метода MCMC. В этой книге мы рассмотрим два способа семплирования: схему Гиббса и алгоритм Метрополиса — Гастингса (Metropolis — Hastings, MH).

Вариационный вывод (variational inference, VI) — это подход к аппроксимации апостериорного распределения $p(x)$, основанный на оптимизации. Здесь мы упрощаем обозначения, придавая общему распределению $p(x)$ смысл апостериорного распределения. Основная идея VI заключается в том, чтобы выбрать приближенное распределение $q(x)$ из семейства удобных (tractable) распределений, а затем сделать это приближение как можно более близким к истинному апостериорному распределению $p(x)$. Под *удобным распределением* понимается всего лишь то, которое легко вычислить. Как мы увидим в разделе книги, посвященном среднему полю (mean-field), приближенная функция $q(x)$ может принимать вид полностью разложенного представления совместного апостериорного распределения. Такая факторизация значительно ускоряет вычисления.

Далее мы введем дивергенцию Кульбака — Лейблера (Kullback — Leibler, KL) и используем ее для измерения близости нашего приближенного распределения к истинному апостериорному. На рис. 1.7 показаны два варианта KL-дивергенции:

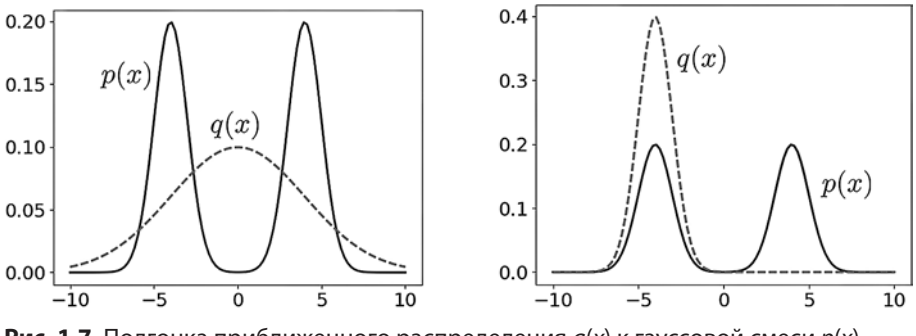

Рис. 1.7. Подгонка приближенного распределения $q(x)$ к гауссовой смеси $p(x)$ с использованием прямой (слева) и обратной дивергенции KL (справа)

Исходное распределение $p(x)$ представляет собой бимодальное гауссово распределение (оно же гауссова смесь с двумя компонентами), в то время как

приближенное распределение $q(x)$ является унимодальным гауссовым распределением. Как показано на рис. 1.7, распределение с двумя пиками мы можем аппроксимировать либо в центре с помощью $q(x)$, которое имеет высокую дисперсию, чтобы отразить наличие бимодального распределения, либо в одной из его мод (правая часть рис. 1.7). Это следует из определений прямой и обратной KL-дивергенции, данных в формуле 1.7. Как мы увидим в последующих главах, путем минимизации KL-дивергенции мы фактически сводим VI к задаче оптимизации.

$$KL(p \| q) = \sum p(x) \log \frac{p(x)}{q(x)} \qquad KL(q \| p) = \sum q(x) \log \frac{q(x)}{p(x)}. \tag{1.7}$$

1.4.2. Современные алгоритмы глубокого обучения

За годы своего существования глубокие нейронные сети прошли путь от базовых блоков сверточной нейросети LeNet до архитектуры визуальных трансформеров. Стали появляться определенные элементы, такие как остаточные связи (residual connections) в модели ResNet, которые стали стандартным решением для современных нейронных сетей произвольной глубины. В последующих главах этой книги мы познакомимся с современными алгоритмами глубокого обучения, в том числе с трансформерами на основе механизма самовнимания (self-attention), с генеративными моделями, такими как вариационные автоэнкодеры, и нейронными сетями графовой структуры.

Мы также обсудим амортизированный вариационный вывод — интересную область исследований, сочетающую в себе выразительные возможности и способность к обучению представлениям, которыми обладают DNN, со знанием предметной области вероятностных графовых моделей. Мы познакомимся с одним из вариантов использования сетей смешанной плотности, где мы будем использовать нейронную сеть для отображения пространства наблюдения на параметры приближенного апостериорного распределения.

Большинство моделей глубокого обучения относятся к категории узкого искусственного интеллекта (narrow AI), демонстрирующего хорошие показатели на конкретном наборе данных. Хотя умение хорошо справляться с узким кругом задач — полезное свойство, мы хотели бы сделать необходимые преобразования, чтобы перейти от узкого ИИ к общему искусственному интеллекту (AGI).

1.5. Реализация алгоритмов

Ключевой частью изучения алгоритмов с нуля является программная реализация. Важно писать хороший код, который не только эффективно использует структуры данных, но также обладает низкой алгоритмической сложностью. Во всех главах мы будем группировать функциональные аспекты кода по классам

и реализовывать различные вычислительные методы с нуля. Тем самым вы познакомитесь с большим количеством методов объектно-ориентированного программирования (ООП), широко используемых в популярных библиотеках ML, в частности в scikit-learn. Хотя целью этой книги является написание всех примеров кода с нуля, мы все же будем иногда пользоваться библиотеками ML (той же scikit-learn,https://scikit-learn.org/stable/), в случае, если таковые имеются, для проверки результатов нашей реализации. На протяжении всей этой книги мы будем использовать язык Python.

1.5.1. Структуры данных

Структура данных — это способ хранения и систематизации данных. Каждая из них обладает своими особенностями в отношении производительности; и поэтому одни структуры больше подходят для данной конкретной задачи, чем другие. В нашей реализации алгоритмов ML мы преимущественно будем использовать *линейные структуры данных*, такие как массивы фиксированного размера, поскольку время доступа к элементу в таком массиве постоянно — $O(1)$. Мы также часто будем использовать динамически изменяемые массивы (например, списки Python), чтобы хранить данные на протяжении серии итераций.

Мы также будем постоянно пользоваться *нелинейными структурами данных* вроде ассоциативных массивов (словарей) и множеств, потому что они построены на основе самобалансирующихся деревьев бинарного поиска (binary search trees, BST), которые гарантируют операции вставки, поиска и удаления со сложностью $O(n\log n)$. И наконец, хеш-таблица, или неупорядоченный ассоциативный массив, — это еще одна широко используемая эффективная структура данных, которая нам пригодится. Она характеризуется временем доступа $O(1)$ при условии отсутствия коллизий.

1.5.2. Парадигмы решения задач

Большинство алгоритмов ML, описанных в этой книге, можно отнести к одной из четырех парадигм решения задач: полный поиск, жадные алгоритмы, «разделяй и властвуй» и динамическое программирование. *Полный поиск* — это метод, в котором решение задачи находится путем обхода всего пространства поиска. Примером задачи машинного обучения, в которой выполняется полный поиск, является точный вывод (exact inference) с помощью полного перечисления. Во время точного вывода необходимо полностью указать набор таблиц вероятностей для выполнения вычислений.

Жадный алгоритм на каждом шаге выбирает локально оптимальный вариант в надежде в итоге прийти к глобально оптимальному решению. Такие алгоритмы часто полагаются на «жадную» эвристику. Примером использования является размещение датчиков. Если, например, имеется помещение и некоторое

количество датчиков температуры, то задача состоит в том, чтобы разместить их таким образом, чтобы максимально увеличить охват помещения.

«Разделяй и властвуй» — это метод, который разбивает проблему на более мелкие, независимые подзадачи, а затем объединяет все их решения. Примером является алгоритм дерева решений CART. Как мы увидим в следующей главе, в CART оптимальный порог для разделения дерева решений определяется путем оптимизации цели классификации (например, индекса Джини). Та же процедура применяется к дереву глубиной на единицу больше, что приводит к рекурсивному алгоритму.

Последняя группа алгоритмов, *динамическое программирование* (ДП), — это метод, который разбивает задачу на более мелкие, перекрывающиеся подзадачи, вычисляет решение для каждой из них и сохраняет его в таблице ДП. Примером может служить обучение с подкреплением (reinforcement learning, RL) при поиске решения уравнений Беллмана. Для небольшого числа состояний мы можем вычислить Q-функцию в табличном виде, используя динамическое программирование.

Итоги

- Алгоритм — это последовательность шагов, необходимых для выполнения определенной задачи. Алгоритмы машинного обучения можно разбить на следующие категории: обучение с учителем, обучение без учителя и глубокое обучение.
- Досконально понимание алгоритмов поможет выбрать наиболее подходящий для решения поставленной задачи, объяснить результаты, устранить сложные проблемы и повысить производительность существующих моделей.
- Байесовский вывод позволяет обновлять наши убеждения о мире на основе наблюдаемых данных, в то время как модели глубокого обучения изучают представления о мире с помощью алгоритма обратного распространения, который минимизирует целевую функцию. Существует два вида байесовского вывода: марковская цепь Монте-Карло (MCMC) и вариационный вывод (VI). Эти два вида сосредоточены вокруг семплирования и аппроксимации апостериорного распределения соответственно.
- Важно писать хороший код, который не только эффективен в использовании структур данных, но и обладает низкой алгоритмической сложностью. Большинство алгоритмов ML, описанных в этой книге, можно отнести к одной из четырех парадигм решения задач: полный поиск, жадные алгоритмы, «разделяй и властвуй» и динамическое программирование.