1 Обзор машинного и глубокого обучения

В этой главе

- ✓ Знакомство с машинным и глубоким обучением.
- ✓ Простая модель машинного обучения — кошачий мозг.
- ✓ Знакомство с глубокими нейронными сетями.

Глубокое обучение преобразило как отдельные сферы — компьютерное зрение, обработка естественного языка и синтез речи, — так и всю область искусственного интеллекта в целом. Из набора разнородных трюков, ни один из которых не давал удовлетворительных результатов в реальных задачах, искусственный интеллект превратился в мощный инструмент решения практических проблем, с которыми сталкивается промышленность. Это не что иное, как революция, творящаяся прямо у нас на глазах. Чтобы оказаться на переднем крае этой революции, важно понимать базовые принципы и абстракции, а не просто запоминать последовательности шагов, описываемые в практических руководствах. Вот тут-то и вступает в дело математика.

В первой главе мы представим обзор глубокого обучения. При этом будем оперировать некоторыми концепциями, разъясняемыми в последующих главах. Не волнуйтесь, если к концу этой главы у вас останутся неразрешенные вопросы, ее цель — подготовить ваш разум к изучению этой сложной темы. Когда при чтении последующих глав вы начнете понимать отдельные концепции, возвращайтесь и перечитайте эту главу.

1.1. ПЕРВОЕ ЗНАКОМСТВО С МАШИННЫМ/ГЛУБОКИМ ОБУЧЕНИЕМ: СМЕНА ПАРАДИГМЫ В ВЫЧИСЛЕНИЯХ

Принятие решений и/или прогнозирование являются одними из ключевых потребностей современной жизни. Под этим подразумеваются прием набора сенсорных или информационных сигналов (входов) и их обработка для выработки решений или оценок.

Например, мозг кошки часто пытается выбрать между следующими вариантами: *убежать* от объекта, находящегося перед ней, *игнорировать* объект или *приблизиться* к нему и помурлыкать. Мозг кошки принимает это решение, обрабатывая сенсорные сигналы, такие как воспринимаемая твердость, острота объекта перед ней и т. д. Это пример проблемы классификации, где выходом является одно из множества возможных решений.

Вот еще несколько примеров задач классификации из жизни.

- *Купить*, *оставить на хранении* или *продать* определенную акцию, основываясь на таких данных, как *история цен на эту акцию* и *изменение цены за последнее время*.
- Распознавание объектов по изображению.
  - Это машина или жираф?
  - Это человек?
  - Это неодушевленный или живой объект?
  - Распознавание лиц — это Том, Дик, Мэри, Эйнштейн или Месси?
- Распознавание действий по видео.
  - Этот человек бежит?
  - Этот человек что-то подбирает?
  - Этот человек совершает насильственное действие?
- Обработка естественного языка (Natural Language Processing, NLP) в цифровых документах.
  - Эта статья относится к сфере политики или спорта?
  - Соответствует ли этот запрос определенной статье в архиве?

Иногда жизнь требует *количественной* оценки вместо классификации. Мозгу льва необходимо оценить, как далеко прыгнуть, чтобы приземлиться на добычу, опираясь на такие входные данные, как скорость добычи и расстояние до нее. Другой пример количественной оценки — оценка стоимости дома на основе таких входных данных, как текущий доход его владельца, статистика преступности в районе и т. д. Алгоритмы, которые вычисляют количественные оценки, называются *регрессорами*.

Вот несколько примеров количественных оценок, необходимых в повседневной жизни.

- Местоположение объекта на изображении — определение координат и размеров минимального прямоугольника, ограничивающего объект.
- Прогнозирование цены акций на основе колебаний цен в прошлом и других мировых событий.
- Оценка схожести пары документов.

Иногда классификация может выполняться на основе количественной оценки. Например, мозг кошки может комбинировать входные данные (твердость, острота и т. д.), чтобы получить количественную оценку угрозы. Если оценка угрозы высока, кошка убегает. Если оценка угрозы близка к нулю, то кошка игнорирует объект. Если оценка угрозы отрицательна, то кошка приближается к объекту и мурлычет.

Многие из этих примеров показаны на рис. 1.1. Во всех случаях машина, то есть мозг, преобразует сенсорные или информационные сигналы в решения или количественные оценки. Машинное обучение стремится подражать этой машине.

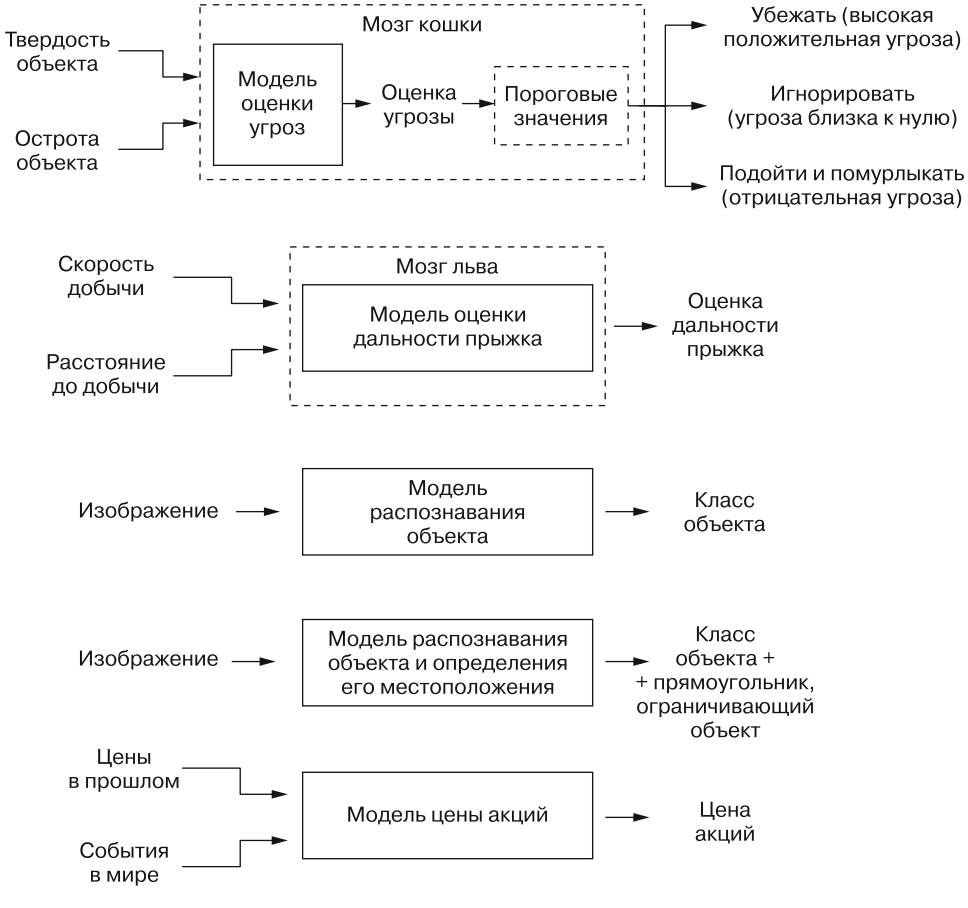

Рис. 1.1. Примеры принятия решений и вычисления количественных оценок в жизни

Обратите внимание, что машине предстоит пройти долгий путь обучения, прежде чем она сможет догнать человеческий мозг. Мозг человека способен в одиночку справиться с тысячами, если не миллионами таких задач. В то же время на нынешнем этапе развития машинного обучения едва ли можно создать универсальную

машину, способную вычислять и принимать широкий спектр оценок и решений. Пока мы пытаемся создавать машины для решения отдельных задач, например для прогнозирования доходности акций или оценки автомобилей. Вы можете спросить: «Подождите-ка! А разве преобразование входных данных в выходные — это не то, чем компьютеры занимаются последние тридцать с лишним лет? Что это за смена парадигмы, о которой вы тут пишете?» Ответ прост: это действительно смена парадигмы, потому что мы не даем машине пошаговых инструкций (то есть программы), как преобразовывать входные данные в выходные, а разрабатываем математическую модель задачи.

Проиллюстрируем эту концепцию на примере. Для простоты рассмотрим гипотетический мозг кошки, которому нужно принять только одно решение в жизни: *убежать от объекта, оказавшегося перед ней, проигнорировать его или приблизиться к нему и замурлыкать*. Это решение является выходом модели, которую мы обсудим. В этом простеньком примере решение принимается на основе всего двух количественных входных данных (их еще называют признаками): кажущейся твердости и остроты объекта (см. рис. 1.1). Мы *не* даем никаких пошаговых инструкций, таких как «если острота больше некоторого порогового значения, то убегай». Вместо этого пытаемся определить *параметризованную* функцию, которая принимает входные данные и преобразует их в желаемое решение или оценку. Простейшая такая функция — взвешенная сумма входных данных:

$$y\,(\text{твердость, острота}) = w_0 \cdot \text{твердость} + w_1 \cdot \text{острота} + b.$$

Весовые коэффициенты w_0, w_1 и смещение b — это параметры функции. Выход y можно интерпретировать как оценку угрозы. Если оценка угрозы превышает пороговое значение, то кошка убегает. Если она близка к 0, то кошка игнорирует объект. Если оценка угрозы отрицательна, то кошка приближается и мурлычет. Для более сложных задач используются более сложные функции.

Обратите внимание на то, что изначально весовые коэффициенты (или просто веса) неизвестны — их нужно оценить. Это делается посредством так называемого обучения модели.

В общем случае решение задачи с помощью машинного обучения состоит из следующих этапов.

- Проектируется параметризованная функция модели (например, взвешенная сумма) с неизвестными параметрами (весами). Она образует *архитектуру модели*. Выбор правильной архитектуры модели во многом определяется опытом инженера по машинному обучению.
- Затем посредством обучения модели получаются оценки весов.
- После оценки весов мы получаем готовую *модель*. Она может принимать произвольные входные данные, не обязательно встречавшиеся ей ранее, и генерировать выходные данные. Процесс обработки произвольных входных данных обученной моделью и получения выходных данных называется *прогнозированием*. Часто также используется термин *инференс*.

В самой популярной разновидности машинного обучения, называемой *обучением с учителем*, заранее готовятся данные, которые включают *примеры входных*

элементов и соответствующие им результаты[1]. Данные для обучения часто создаются вручную: человек просматривает каждый входной элемент и определяет желаемый (он же целевой) результат. Обычно это самый трудоемкий этап в машинном обучении.

Вот некоторые возможные элементы обучающих данных для примера с мозгом кошки.

- Вход: (*твердость* = 0,01, *острота* = 0,02) → угроза = –0,9 → *решение*: «подойти и помурлыкать».
- Вход: (*твердость* = 0,5, *острота* = 0,6) → угроза = 0,01 → *решение*: «игнорировать».
- Вход: (*твердость* = 0,99, *острота* = 0,97) → угроза = 0,9 → *решение*: «убежать».

Здесь предполагается, что входные значения твердости и остроты находятся в диапазоне от 0 до 1.

Что происходит во время обучения? Мы итеративно обрабатываем элементы входных обучающих данных. Для каждого элемента известен желаемый (целевой) выход, и на каждой итерации мы корректируем значения весов модели так, чтобы выход функции модели для этого конкретного элемента входных данных стал хотя бы немного ближе к соответствующему целевому выходу. Предположим, что на данной итерации значения весов $w_0 = 20$, $w_1 = 10$ и $b = 50$. Для входного элемента (*твердость* = 0,01, *острота* = 0,02) мы получаем выходную оценку угрозы $y = 50{,}3$, сильно отличающуюся от желаемого значения $y = -0{,}9$. Скорректируем веса, например, уменьшим смещение: $w_0 = 20$, $w_1 = 10$ и $b = 40$. После этого оценка угрозы $y = 40{,}3$ остается все еще слишком далекой от желаемого значения, но уже немного ближе к нему. Проделав то же самое для множества элементов обучающих данных, мы добьемся того, что веса приблизятся к идеальным значениям. Обратите внимание, что здесь не обсуждается, как определить величину корректировки весов, — для этого требуются более глубокие математические расчеты, которые мы рассмотрим позже.

Как отмечалось ранее, итеративный процесс настройки весов называется *обучением* или *подгонкой*. В начале обучения веса имеют случайные значения, поэтому выходные данные часто не соответствуют желаемым. Но с каждой итерацией машина учится генерировать правильный выход. И когда у нее это получается, модель готова к развертыванию в реальном мире. На этапе прогнозирования обученная модель, получив произвольные входные данные, выдаст что-то близкое к желаемому выходу.

Вероятно, именно так и работает живой мозг. Он содержит эквиваленты математических моделей для различных задач. Здесь веса — это прочность связей (иначе называемых синапсами) между различными нейронами в мозге. Сначала параметры не настроены и мозг постоянно ошибается. Например, мозг младенца часто ошибается при идентификации съедобных объектов — любой, у кого был ребенок,

---

[1] Если вам доводилось заниматься машинным обучением, то вы уже поняли, что здесь мы говорим здесь об обучении с учителем. Существуют также машины, которым не нужны известные результаты для обучения, — так называемые машины, обучающиеся без учителя. О них мы поговорим позже.

поймет, о чем мы говорим. Но каждая попытка настраивает параметры (поедание бело-зеленых прямоугольных бумажек со значком $ на них вызывает возмущение родителей, а значит, в будущем их не следует есть и т. д.). В конце концов машина настраивает свои параметры и начинает получать более точные результаты.

Здесь следует отметить один тонкий момент. Во время обучения машина настраивает параметры модели так, чтобы получать желаемый результат, но может ориентироваться *только на входные обучающие данные*. То есть во время обучения она видит лишь малую часть всех возможных входных данных — мы *не* строим таблицу соответствия известных входных данных известным выходным данным. Следовательно, когда машина оказывается в реальном мире, она обрабатывает преимущественно входные данные, которых прежде никогда не видела. Можем ли мы гарантировать, что она выдаст правильный результат, получив никогда не встречавшиеся ей данные? Честно говоря, нет. Однако в большинстве реальных задач входные данные на самом деле неслучайны. Они подчиняются некоторым закономерностям. Мы надеемся, что во время обучения машина увидит достаточно данных, чтобы уловить эту закономерность. Тогда при новых входных данных ее выход будет близок к желаемому. Чем ближе распределение обучающих данных к реальности, тем выше вероятность получить правильный результат.

1.2. ВЗГЛЯД НА МАШИННОЕ ОБУЧЕНИЕ КАК НА АППРОКСИМАЦИЮ ФУНКЦИЙ: МОДЕЛИ И ИХ ОБУЧЕНИЕ

Как отмечалось в разделе 1.1, для создания машины, подобной мозгу, которая выполняет классификацию или вычисляет оценки, нужно найти математическую функцию (модель), преобразующую входные данные в желаемые результаты. К сожалению, в реальных ситуациях мы обычно не знаем, как выглядит функция преобразования. Например, не знаем, как выглядит функция, которая на основе стоимости акций в прошлом, событий в мире и так далее сможет предсказать, как изменится стоимость акций в будущем (и это незнание мешает нам построить систему оценки акций и разбогатеть). У нас есть лишь данные для обучения — набор входных данных, для которых известен результат. Как тут быть? Ответ: можно попробовать смоделировать неизвестную функцию, то есть создать функцию, которая сможет служить заменой неизвестной функции. С этой точки зрения машинное обучение — не что иное, как аппроксимация функции: мы просто пытаемся аппроксимировать неизвестную функцию классификации или вычисления оценки.

Давайте кратко повторим основные идеи из предыдущего раздела. С помощью машинного обучения мы пытаемся решать задачи, которые можно абстрактно рассматривать как преобразование набора входных данных в выходные. Выходные данные — это либо класс, либо значение оценки. Не зная истинной функции преобразования, мы пытаемся придумать аппроксимирующую ее функцию. Начинаем с проектирования, используя имеющееся физическое понимание задачи — модельной функции с настраиваемыми параметрами, которая может служить представлением истинной функции. Функция — это *архитектура модели*,

а настраиваемые параметры называют *весами*. В простейшей архитектуре модели выходные данные представляют взвешенную сумму входных значений. Архитектура модели не полностью определяет модель, кроме нее, необходимо еще определить фактические значения параметров (весов). Для достижения этой цели используется процесс *обучения*. Во время него мы подбираем оптимальный набор весов, преобразующих обучающие входные данные в выходные, которые максимально соответствуют обучающим выходным данным. Затем развертываем эту машину в промышленном окружении: ее веса и функция полностью определены, поэтому она просто применяет функцию к входным данным и генерирует выходные данные. Это называется *выводом* или *прогнозированием*. Конечно, обучающие входные данные составляют лишь часть всех возможных входных данных, поэтому нет никаких гарантий того, что модель будет давать желаемый результат для любых реальных входных данных. Успех модели зависит от уместности выбранной архитектуры, а также от качества и количества обучающих данных.

ПОЛУЧЕНИЕ ОБУЧАЮЩИХ ДАННЫХ

Самой большой проблемой после создания модели машинного обучения является получение данных для обучения. Когда специалисты могут себе это позволить, они привлекают людей для создания обучающего набора вручную (такие наборы целевых выходных данных иногда называют *эталонными* или *истинными данными*, а также используют англоязычное название *ground truth — GT*). Процесс, известный как *разметка* или *курирование человеком*, предполагает привлечение армии людей, которые просматривают огромный объем входных обучающих данных и генерируют соответствующие истинные выходные данные. Для некоторых хорошо изученных задач можно получить обучающие данные из Интернета, а если это невозможно, то подготовка таких данных превращается в сложную задачу. Подробнее об этом мы поговорим позже.

Теперь рассмотрим процесс построения модели на конкретном примере — на модели «мозг кошки», показанной на рис. 1.1.

1.3. ПРОСТАЯ МОДЕЛЬ МАШИННОГО ОБУЧЕНИЯ «МОЗГ КОШКИ»

Для простоты представим себе гипотетическую кошку, которая в своей жизни должна принимать только одно решение: убежать от объекта, оказавшегося перед ней, проигнорировать его или подойти и помурлыкать. Она принимает это решение только на основе двух количественных входных данных, характеризующих объект (см. рис. 1.1).

ПРИМЕЧАНИЕ

Эта глава представляет собой упрощенный обзор машинного/глубокого обучения. Ее содержание опирается на некоторые математические концепции, которые будут представлены позже, однако рекомендуем прочитать эту главу сейчас и, если потребуется, перечитать ее после прочтения глав о векторах и матрицах.

1.3.1. Входные признаки

Мы имеем два входных признака — x_0, обозначающий *твердость*, и x_1, обозначающий *остроту*. Мы можем *нормализовать* входные данные без потери общности. Это довольно популярный трюк, когда входные значения в диапазоне от минимально возможного значения $v_{\min}$ до максимально возможного значения $v_{\max}$ преобразуются в значения от 0 до 1. Для преобразования произвольного входного значения v в нормализованное значение v_{norm} используется формула:

$$v_{\text{norm}} = \frac{(v - v_{\min})}{(v_{\max} - v_{\min})}. \tag{1.1}$$

На математическом языке преобразование с помощью уравнения (1.1), $v \in [v_{\min}, v_{\max}] \rightarrow v_{\text{norm}} \in [0, 1]$, отображает значения v из входного диапазона $[v_{\min}, v_{\max}]$ в выходные значения v_{norm} в диапазоне $[0, 1]$.

Двухэлементный вектор $\vec{x} = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \in [0, 1]^2$ кратко представляет один входной экземпляр.

1.3.2. Выходные решения

Окончательный результат может принимать одно из трех возможных значений: 0, подразумевающее бегство от объекта, 1, подразумевающее игнорирование объекта, и 2, подразумевающее приближение к объекту и мурлыканье. В машинном обучении класс можно вычислить напрямую. Однако в этом примере наша модель должна оценить степень угрозы. Она интерпретируется следующим образом: угроза высокая положительная = убежать, угроза близка к нулю = игнорировать, угроза высокая отрицательная = приблизиться и помурлыкать (отрицательная угроза привлекательна).

Мы можем принять окончательное решение убежать/игнорировать/подойти на основе оценки угрозы, сравнив уровень угрозы y с пороговым значением δ:

$$y \begin{cases} > \delta \rightarrow \text{высокая угроза, убежать;} \\ \geq -\delta \ u \leq \delta \rightarrow \text{угроза близка к нулю, игнорировать;} \\ < -\delta \rightarrow \text{отрицательная угроза, подойти и помурлыкать.} \end{cases} \tag{1.2}$$

1.3.3. Аппроксимация модели

Теперь мы должны сделать самый важный шаг — оценить функцию, преобразующую входной вектор в выходной. Немного злоупотребив терминами, обозначим эту функцию, а также выходной вектор как y. В математической нотации мы должны оценить $y(\vec{x})$.

Конечно, мы не знаем, как выглядит идеальная функция. Но попытаемся аппроксимировать эту неизвестную функцию, основываясь на обучающих данных. Это делается в два этапа.

1. *Этап выбора архитектуры модели* — проектирование параметризованной функции, которая, как ожидается, послужит хорошим заменителем неизвестной идеальной функции.

2. *Этап обучения* — подбор таких параметров функции, при которых результат преобразования входных обучающих данных будет максимально точно соответствовать выходным обучающим данным.

1.3.4. Выбор архитектуры модели

На этом этапе применяются разные подходы к машинному обучению. В примере моделирования мозга кошки мы используем простейшую возможную модель. Она имеет три параметра: w_0, w_1 и b. Их можно компактно представить с помощью вектора с двумя элементами $\vec{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \in \mathbb{R}^2$ и постоянного смещения $b \in \mathbb{R}$ (здесь $\mathbb{R}$ обозначает множество всех действительных чисел, $\mathbb{R}^2$ — множество двумерных векторов с действительными элементами и т. д.). Модель выдает оценку угрозы y, которая вычисляется так:

$$y(x_0, x_1) = w_0 x_0 + w_1 x_1 + b = \begin{bmatrix} w_0 & w_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} + b = \vec{w}^{\mathrm{T}} \vec{x} + b. \tag{1.3}$$

Обратите внимание на то, что b — особый параметр. Это константа, которая не умножается ни на один из входов. В машинном обучении этот параметр принято называть *смещением*. В отличие от него другие параметры умножаются на входы как веса.

1.3.5. Обучение модели

Выбор архитектуры модели точно определяет, какая параметрическая функция будет использоваться для моделирования неизвестной функции $y(\vec{x})$ и преобразования входных данных в выходные. Но нужно еще оценить параметры функции. Итак, у нас есть функция с неизвестными параметрами и мы должны подобрать их, опираясь на набор входных данных с известными выходными данными (обучающие данные). Сделаем это так, чтобы выходные данные, получаемые в ответ на ввод обучающих входных данных, были максимально близки к соответствующим известным выходным данным.

ИТЕРАТИВНОЕ ОБУЧЕНИЕ

Эта задача хорошо изучена и в математике известна как задача *подгонки функции*. Однако с появлением машинного обучения ее масштаб изменился. В машинном обучении мы имеем дело с обучающими данными, содержащими миллионы и миллионы элементов. Это изменило философию решения. Математики привыкли использовать решения в *аналитической форме*, где параметры оцениваются путем прямого решения уравнений, включающих *все* элементы обучающих данных. В машинном обучении применяются итеративные подходы, когда за раз обрабатывается *несколько* элементов обучающих данных или даже один элемент. При итеративном подходе нет необходимости хранить в памяти компьютера сразу все обучающие данные. Их можно загружать небольшими порциями и обрабатывать по частям. Мы проиллюстрируем этот подход на примере с мозгом кошки.

Если говорить конкретно, то целью процесса обучения является подгонка параметров w_0, w_1 и b, или, что одно и то же, вектора $\vec{w}$ с константой b из уравнения (1.3), до такой степени, что результат функции $y(x_0, x_1)$, получающей на входе обучающие данные (x_0, x_1), будет настолько точно соответствовать известным целевым (или истинным) данным в обучающем наборе, насколько это возможно.

Пусть обучающие данные состоят из $N + 1$ входов $\vec{x}^{(0)}, \vec{x}^{(1)}, \ldots, \vec{x}^{(N)}$, где каждый элемент $\vec{x}^{(i)}$ — это вектор 2×1, представляющий один экземпляр входных обучающих данных. Соответствующие желаемые оценки угроз (выходы) — это $y_{\text{gt}}^{(0)}, y_{\text{gt}}^{(1)}, \ldots, y_{\text{gt}}^{(N)}$ (здесь нижний индекс gt означает *ground truth* — «истинные данные»). Соответственно, мы можем сказать, что обучающие данные состоят из $N + 1$ пар входных и выходных данных:

$$\left(\vec{x}^{(0)}, y_{\text{gt}}^{(0)}\right), \left(\vec{x}^{(1)}, y_{\text{gt}}^{(1)}\right), \ldots, \left(\vec{x}^{(N)}, y_{\text{gt}}^{(N)}\right).$$

Предположим, что $\vec{w}$ обозначает пока что неизвестные оптимальные параметры модели. Тогда, учитывая произвольные входные данные $\vec{x}$, машина вычислит значение угрозы $y_{\text{predicted}} = \vec{w}^{\mathrm{T}}\vec{x} + b$. Для i-й пары обучающих данных $\left(\vec{x}^{(0)}, y_{\text{gt}}^{(i)}\right)$ она выдаст оценку

$$y_{\text{predicted}}^{(i)} = \vec{w}^{\mathrm{T}}\vec{x}^{(i)} + b$$

для желаемого выхода $y_{\text{gt}}^{(i)}$. Отсюда квадрат ошибки (или функция потерь), допущенной машиной на i-м экземпляре обучающих данных, равен[1]:

$$e_i^2 = \left(y_{\text{predicted}}^{(i)} - y_{\text{gt}}^{(i)}\right)^2.$$

Общая величина потерь для всего набора обучающих данных получается сложением значений функции потерь на отдельных экземплярах:

$$E^2 = \sum_{i=0}^{i=N} e_i^2 = \sum_{i=0}^{i=N} \left(y_{\text{predicted}}^{(i)} - y_{\text{gt}}^{(i)}\right)^2 = \sum_{i=0}^{i=N} \left(\vec{w}^{\mathrm{T}}\vec{x}_i + b - y_{\text{gt}}^{(i)}\right)^2.$$

Цель обучения — найти параметры (веса) $\vec{w}$ модели, минимизирующие общую ошибку E. Как именно это делается, будет описано позже.

В большинстве случаев невозможно найти аналитическое решение для определения оптимальных $\vec{w}$ и b, поэтому мы используем итеративный подход, представленный в алгоритме 1.1.

На входе в этот алгоритм мы имеем случайные значения параметров и продолжаем настраивать их так, чтобы хоть немного уменьшить общую ошибку. Продолжаем итерации до тех пор, пока ошибка не станет достаточно маленькой.

В математическом смысле итерации продолжаются до тех пор, пока ошибка не станет минимальной. Но на практике мы обычно останавливаем их, когда результаты окажутся достаточно точными для решаемой задачи. Стоит еще раз

---

[1] Следует отметить, что величину ошибки/функции потерь принято возводить в квадрат, чтобы сделать ее независимой от знака. Если мы хотим получить на выходе, скажем, число 10, то будем одинаково рады или не рады, если на выходе окажется число 9,5 или 10,5. То есть ошибки +5 и –5 фактически равноценны, поэтому мы стремимся сделать величину ошибки независимой от знака.

подчеркнуть, что под ошибкой понимается только отклонение от известных значений в обучающих данных.

Алгоритм 1.1. Обучение модели с учителем

Инициализировать параметры $\vec{w}, b$ случайными значениями

- повторять итерации, пока ошибка не станет достаточно маленькой

while $(E^2 = \sum_{i=0}^{i=N} \left(\vec{w}^{\mathrm{T}} \vec{x}_i + b - y_{\mathrm{gt}}^{(i)}\right)^2 > \textit{порогового значения})$ **do**

- выполнить обход всех экземпляров обучающих данных

for $\forall_i \in [0, N]$ **do**

- подробности вы найдете в разделе 3.3 после знакомства с градиентами

Скорректировать $\vec{w}, b$ так, чтобы уменьшить E^2

end for

end while

- запомнить полученные значения параметров как оптимальные

$\vec{w}_* \leftarrow \vec{w}, b_* \leftarrow b$

1.3.6. Прогнозирование

Наконец обученная модель (с оптимальными параметрами $\vec{w}_*, b_*$) развертывается в промышленном окружении. Она получит новые входные данные $\vec{x}$ и выдаст результат $y_{\mathrm{predicted}}(\vec{x}) = \vec{w}_*^{\mathrm{T}} \vec{x} + b_*$. Классификация выполняется путем сравнения с пороговым значением $y_{\mathrm{predicted}}$, как показано в уравнении 1.2.

1.4. ГЕОМЕТРИЧЕСКИЙ ВЗГЛЯД НА МАШИННОЕ ОБУЧЕНИЕ

Каждый элемент входных данных модели мозга кошки представлен массивом из двух чисел, x_0 (обозначает твердость объекта) и x_1 (обозначает остроту объекта), или, что эквивалентно, вектором $\vec{x}$ размером 2×1. Такой вектор можно представить как точку в многомерном пространстве. Входное пространство часто называют *пространством признаков* — пространством всех характерных признаков, которые должна исследовать модель. В нашем случае размерность пространства признаков равна двум, но в реальных задачах она может исчисляться сотнями, тысячами и даже бóльшим количеством измерений. Точная размерность входных данных меняется от задачи к задаче, но интуитивное представление в виде точки остается неизменным.

Выход y тоже следует рассматривать как точку в другом многомерном пространстве. В данной задаче размерность выходного пространства равна единице, в реальных задачах может быть намного больше, но чаще всего она меньше размерности пространства признаков.

С геометрической точки зрения модель машинного обучения отображает точку из пространства признаков в точку в выходном пространстве. Предполагается, что классификацию или вычисление оценки моделью проще выполнить в выходном пространстве, чем в пространстве признаков. В частности, *в задаче классификации входные точки, принадлежащие отдельным классам, будут отображаться в обособленные кластеры в выходном пространстве*.

Продолжим рассматривать пример с моделью мозга кошки, чтобы проиллюстрировать идею. Как было сказано ранее, наше пространство признаков двумерное, с осями координат X_0, обозначающей твердость, и X_1, обозначающей остроту[1]. Отдельные точки в этом двумерном пространстве обозначены строчными буквами (x_0, x_1) (рис. 1.2). Как показано на диаграмме, смоделировать оценку угрозы можно как расстояние от линии $x_0 + x_1 = 1$.

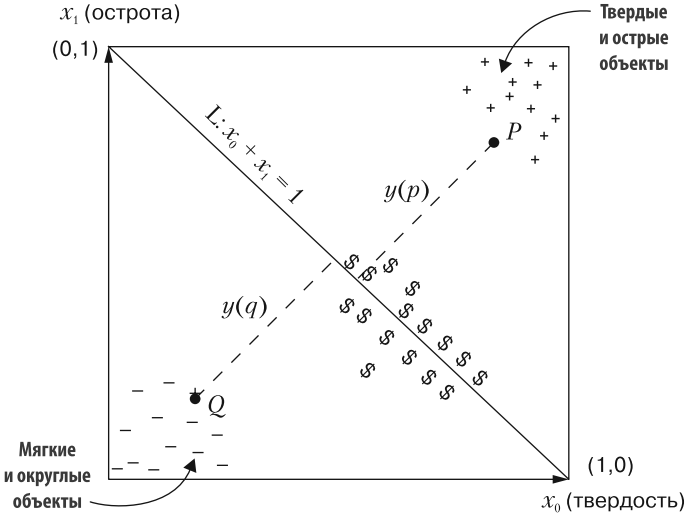

Рис. 1.2. Двумерное пространство входных точек для модели мозга кошки. В нижнем левом углу показаны объекты с низкой твердостью и низкой остротой (–), а в верхнем правом углу — с высокой твердостью и высокой остротой (+). Промежуточные значения находятся около диагонали ($)

Как мы знаем из геометрии, на двумерной плоскости с осями координат X_0 и X_1 расстояние от точки (a, b) до прямой $x_0 + x_1 = 1$ равно $y = \frac{a+b-1}{\sqrt{2}}$. Рассматривая знак y, можно определить, с какой стороны от прямой находится точка. В простой ситуации, изображенной на рис. 1.2, наблюдения показывают, что оценка угрозы может быть выражена знаком расстояния y от диагональной прямой $x_0 + x_1 - 1 = 0$. Чтобы принять решение бежать/игнорировать/приблизиться, мы можем установить порог y. Значения, близкие к нулю, подразумевают игнорирование, положительные значения — убегание, а отрицательные — приближение и мурлыканье. Из курса геометрии нам известно, что расстояние от произвольной точки ($x_0 = a$, $x_1 = b$) до прямой $x_0 + x_1 - 1 = 0$ вычисляется как $\frac{a+b-1}{\sqrt{2}}$. Иначе говоря, функция $y(x_0, x_1) = \frac{x_0 + x_1 - 1}{\sqrt{2}}$ является возможной функцией оценки угрозы для модели мозга кошки. Обучение должно сходиться к $w_0 = 1/\sqrt{2}, w_1 = 1/\sqrt{2}$ и $b = -1/\sqrt{2}$.

---

[1] Для обозначения координат вместо привычных X и Y мы используем X_0 и X_1, чтобы не исчерпать символы при переходе в пространства более высоких размерностей.

Итак, упрощенная модель оценки угрозы выглядит следующим образом:

$$y(x_0, x_1) = \frac{1}{\sqrt{2}} x_0 + \frac{1}{\sqrt{2}} x_1 - \frac{1}{\sqrt{2}}. \tag{1.4}$$

Она отображает двумерные входные точки, обозначающие твердость и остроту объекта, в одномерное значение, соответствующее расстоянию со знаком до разделительной прямой. Это расстояние, интерпретируемое как оценка угрозы, позволяет разделить классы (отрицательная, нейтральная и положительная угроза) с помощью порогового значения, как показано в уравнении (1.2). Классы образуют в выходном пространстве обособленные кластеры, показанные знаками +, – и $. Низкие значения входных данных дают отрицательные оценки угрозы (кошка приближается и мурлычет), например $y(0,\ 0) = -1/\sqrt{2}$. Высокие значения входных данных создают высокие оценки угрозы (кошка убегает), например $y(1,\ 1) = 1/\sqrt{2}$. Средние значения входных данных дают почти нулевую угрозу (кошка игнорирует объект), например $y(0{,}5,\ 0{,}5) = 0$. В такой тривиальной задаче мы могли бы выбрать параметры модели простым наблюдением, но в реальных ситуациях для этого необходимо обучение.

Геометрическая интерпретация сохраняется и в пространствах с бóльшим числом измерений. В общем случае n-мерный входной вектор $\vec{x}$ отображается в m-мерный выходной вектор (обычно $m < n$) так, что в выходном пространстве задача решается намного проще. На рис. 1.3 показан пример с трехмерным пространством признаков.

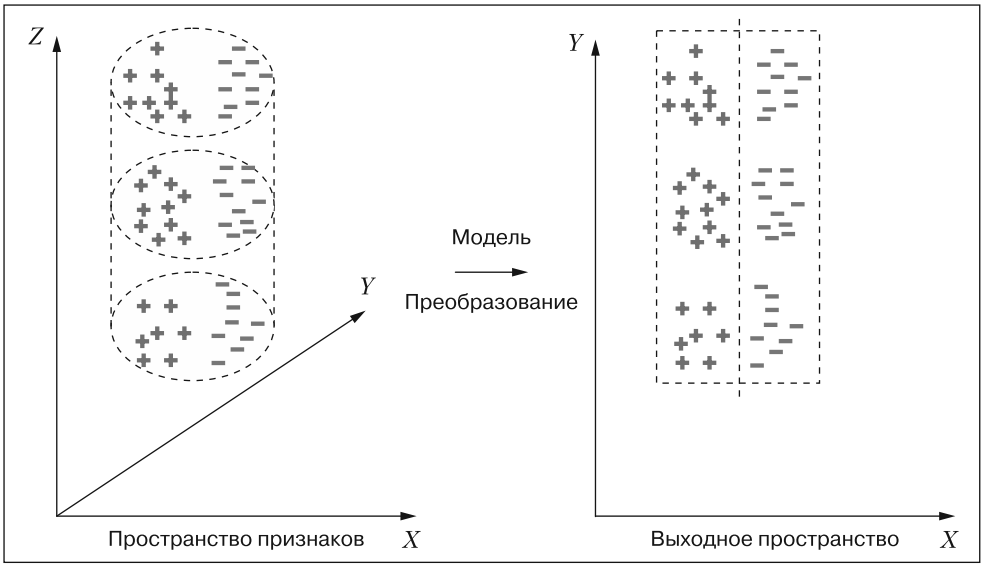

Рис. 1.3. Модель отображает точки из входного пространства (признаков) в выходное пространство, где проще выполнить разделение на классы. Например, здесь входные точки, принадлежащие двум классам, красному (+) и зеленому (–), распределены по объему цилиндра в трехмерном пространстве признаков. Модель разворачивает цилиндр в прямоугольник, и точки отображаются в двумерное плоское выходное пространство, где два класса можно разделить с помощью простого линейного решающего правила