

1. ВВЕДЕНИЕ В МАШИННОЕ ОБУЧЕНИЕ

1.1. Понятие машинного обучения

Машинное обучение изучает методы и алгоритмы решения задач при помощи компьютера, основанные на анализе уже решенных задач и постепенном улучшении качества решения. Для конкретной задачи компьютер изначально не обладает способностью ее решения, но методы машинного обучения позволяют ему этому научиться, то есть можно говорить о «самообучении машин».

Для обучения машины решению задачи необходимо предоставить ей эмпирические данные – *обучающую выборку*, на основе которой вырабатывается *алгоритм решения*. Обучающая выборка представляет собой множество экземпляров различных объектов, каждый из которых также может включать либо не включать в себя ответы.

Где применяются алгоритмы машинного обучения:

- Database mining: получение информации из больших объемов данных (переходы по ссылкам веб-страниц, медицинские истории, биоинформатика, ...)
- Приложения, которые не могут быть хорошо запрограммированы вручную (автономные транспортные средства, распознавание символов, обработка текстов на естественном языке, компьютерное зрение, ...)
- Самообучаемые программы (системы рекомендаций интернет-магазинов, ...)
- Искусственный интеллект (имитация работы мозга, ...)

Пример. Задача кредитного скоринга. Финансовые организации, занимающиеся выдачей кредитов населению, желали бы иметь возможность определять, кому стоит выдавать кредит («хороший клиент»), а кому не стоит, потому что он с большой вероятностью его не вернет («плохой клиент»). Для обучения машины классификации клиентов на «хороших» и «плохих» используется обучающая выборка данных о выданных кредитах за определенных промежутков времени в прошлом. Выборка включает: параметры заемщика (возраст, образование, уровень дохода и т.д.), параметры кредита (сумма, срок и т.д.) и ответ (заемщик был «хорошим» или «плохим»). На основании этой выборки строится алгоритм, который по параметрам нового заемщика и кредита (уже не из обучающей выборки) даст ответ, стоит ли выдавать кредит или нет. ■

Очевидно, что машина может ошибаться в своем решении и давать неверные ответы. Важный вопрос насколько сильно и как часто она ошибается? После одного алгоритма обучения машина может ошибаться сильнее, чем после другого, поэтому говорят о *качестве обучения*. Важной особенностью машинного обучения является то, что на основе эмпирических данных можно качество обучения улучшить.

Что такое машинное обучение?

Машинное обучение – область науки, которая изучает возможность компьютеров обучаться без необходимости их непосредственно программировать (Артур Самюэл, 1959)

Математик Артур Самюэл написал программу игры в шашки, которая обучалась в процессе игры. Программа изучала, какие позиции и ходы на доске ведут к выигрышу, а какие к проигрышу, играя сама против себя. От партии к партии, программа играла все лучше и лучше.

В одной из самых популярных книг по машинному обучению дано следующее определение.

Компьютерная программа *обучается* на основе опыта E по отношению к некоторому классу задач T и меры качества P , если качество решения задач из T , измеренное на основе P , улучшается с приобретением опыта E . [Т.М. Mitchell. Machine Learning. McGraw-Hill, 1997.]

Для задачи игры в шашки задачей T является способность выиграть в шашки, опы-

том E является множество партий, сыгранных против самой себя, качество P можно определить как вероятность выигрыша в партии.

Пример: распознавание рукописных символов, E – примеры рукописного написания символов, T – задача распознавания текста, P – вероятность правильного распознавания символов. Добавляя в обучающую выборку больше различных вариантов написания рукописных символов, можно улучшить вероятность правильного распознавания. ■

В зависимости от того, имеет ли обучающая выборка ответы, задачи машинного обучения разделяют на два больших класса: «обучение с учителем» и «обучение без учителя».

Мы имеем дело с *обучением с учителем*, когда обучающая выборка имеет правильные ответы, и задачей машинного обучения является найти такой алгоритм решения, который будет давать правильные ответы и для объектов за пределами обучающей выборки. Если ответы принадлежат конечному дискретному множеству (например, «хороший» или «плохой» клиент), такую задачу называют *задачей классификации*. Когда ответы лежат в каком-то непрерывном множестве, задача называется *задачей регрессии*.

Если обучающая выборка не имеет ответов, а только известно к какому множеству они принадлежат, говорят об *обучении без учителя*. В данном случае ответы машина находит сама в процессе обучения. Если множество ответов является дискретным, такую задачу называют *задачей кластеризации*.

Рассмотрим несколько примеров.

Пример: предсказание цен на недвижимость, имея данные о площади помещения. На основании прошлых продаж имеется обучающая выборка, включающая в себя площадь помещения в квадратных метрах и цена (в условных единицах, которые мы назовем м.к.) (табл. 1).

Табл. 1.

№ объекта	Площадь, м ²	Цена, м.к.
1	2104	460
2	1416	232
3	1534	315
4	852	178
5	1948	305
6	950	293
7	611	141
8	1751	343
9	451	102
10	1244	209
11	1416	286

Вопрос: какая цена будет у дома площадью 750 м²? Расположим объекты обучающей выборки на графике, по горизонтальной оси площадь помещения, по вертикальной – цена (рис. 1.1).

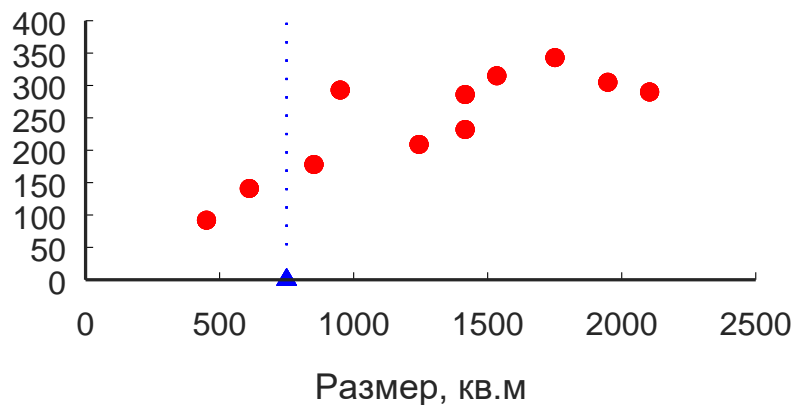
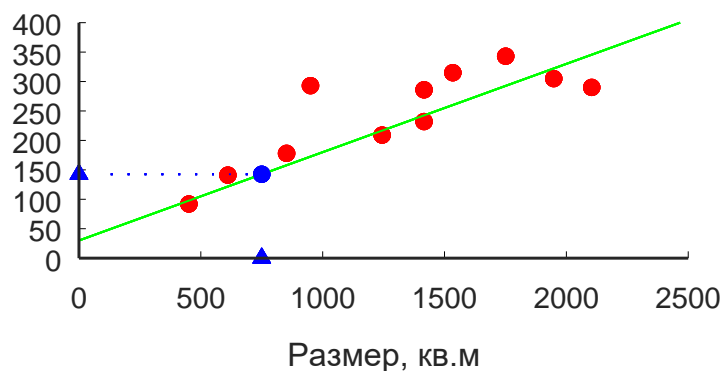
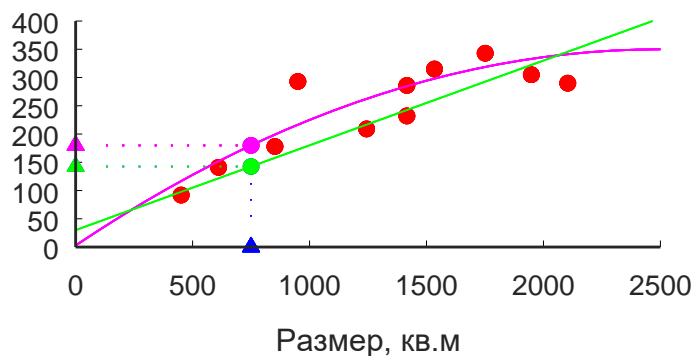


Рис. 1.1

Пример является типичной задачей регрессии, решаемой на основе методов обучения с учителем. В качестве решения проведем прямую линию через скопление точек на плоскости (линейная регрессия), как показано на рис. 1.2, а. Будем считать, что все заранее неизвестные значения функции будут лежать на этой прямой.



а)



б)

Рис. 1.2

В таком случае, расчетная стоимость помещения составляет примерно 150 м.к. Такое решение не является самым лучшим. Например, проведем другую линию (рис. 1.2, б). Хорошо видно, что эта линия находится ближе к точкам обучающей выборки. По второму решению, прогнозируемая стоимость помещения будет примерно 180 м.к. ■

Пример: классификация раковых заболеваний. Дана обучающая выборка (табл. 2), в которой показаны медицинские данные пациентов, больных раком: размер раковой опухоли (в мм) и результат пункции этой опухоли (доброкачественная или злокачественная).

Табл. 2.

Размер опухоли	11	19	21	27	34	38	45	51	59	63
Злокачественная	0	0	0	1	0	1	0	1	1	1

Расположим точки на графике (рис. 1.3, а), по горизонтальной оси отсчитывается размер опухоли, а по вертикальной – только два значения: 0 – если опухоль доброкачественная, 1 – если злокачественная.

Если теперь у нас появится пациент с опухолью размером 24, какая она у него будет, скорее доброкачественная или злокачественная?

Данный пример иллюстрирует задачу классификации, решаемую на основе обучения с учителем.

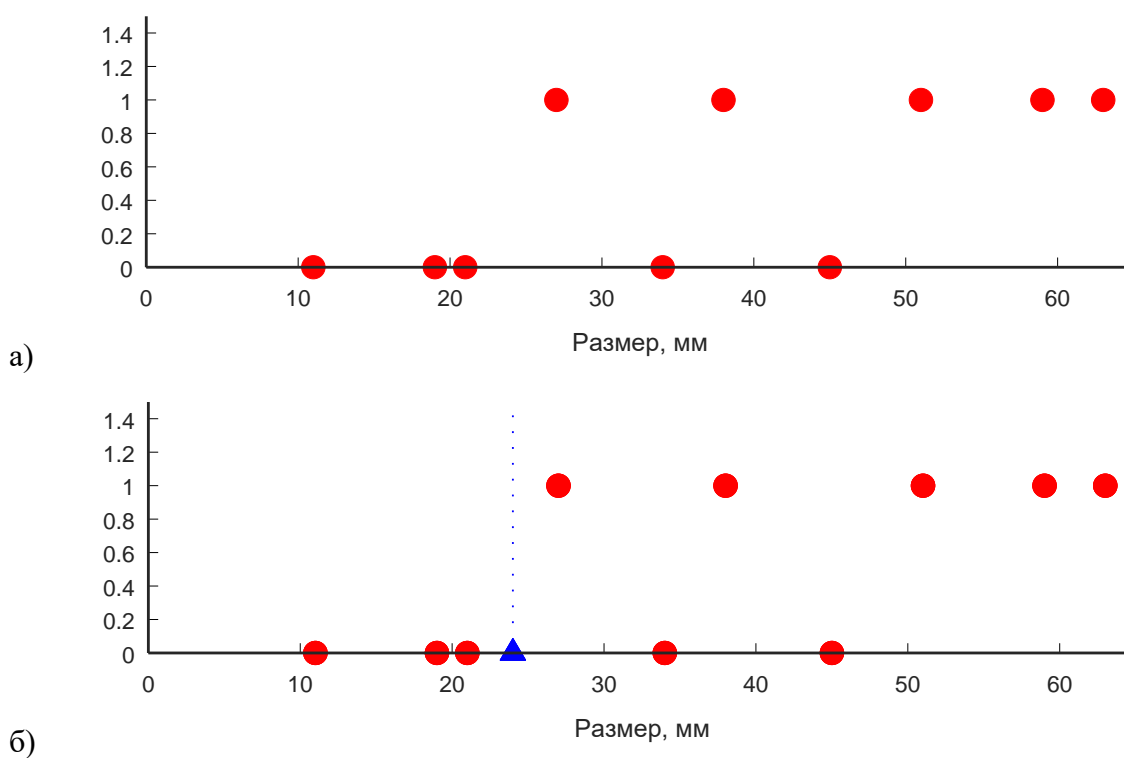


Рис. 1.3

Одно из ее решений может выглядеть следующим образом: будем считать, что опухоль с размерами меньше 35 скорее доброкачественная, чем злокачественная (рис. 1.3, б). Тогда опухоль с размером 24 мы отнесем к доброкачественным. Как видно, слева от 35 встречается случай злокачественной опухоли, как и справа от 35 встречается случай доброкачественной. Решение задачи классификации зачастую не является точным. ■

Пример: Дополним предыдущий пример. Теперь помимо размера опухоли имеется дополнительная информация – возраст пациента (см. табл. 3).

Табл. 3.

Размер опухоли	15	19	27	24	21	9	51	...	24	47
Возраст	20	12	41	45	20	35	20	...	45	52
Злокачественная	0	0	1	0	0	1	1	...	0	1

В этом случае, если мы расположим на графике эти данные, они будут выглядеть, как показано на рис. 1.4.

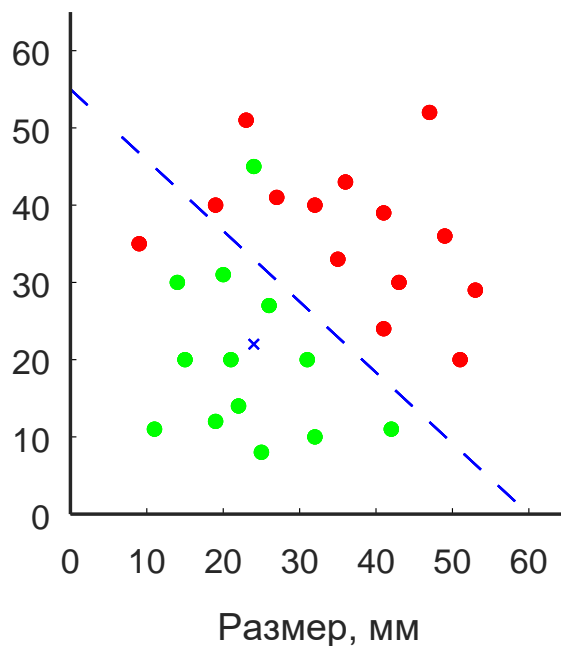


Рис. 1.4

Видно, что если мы проведем линию, которая отделяет большую часть точек, относящихся к пациентам со злокачественной формой от точек пациентов с доброкачественной формой, то мы можем использовать эту линию в качестве критерия классификации. Например, пациент с размером опухоли 24 и возрастом 22 будет классифицирован, как вероятнее всего имеющий доброкачественную форму. ■

Пример: задача информационного поиска. У нас имеется большая подборка новостных интернет-статей, необходимо найти из них статьи, описывающие новости по близким темам (например, новости об изменении цены на нефть, о текущем чемпионате мира по футболу и т.д.).

Данный пример является задачей кластеризации, задачей обучения без учителя. Обучающая выборка не имеет информации о темах новостей, их необходимо определить в процессе обучения. Более того, в будущем, при появлении новых новостных тем, эта классификация автоматически будет дополняться и перестраиваться.

Как правило, такие задачи решаются путем выделения в тексте статей ключевых слов, подсчете их количества и группировке статей с близкими ключевыми словами.

В случае, если бы ключевых слов было всего два (например, слово «мяч» и слово «реформа»), то можно было бы посчитать количество, сколько раз каждое из них встречается в различных статьях, как показано в таблице 3.

Табл. 3

Статья №	Кол-во слов «Мяч»	Кол-во слов «Реформа»
1	6	3
2	8	2
3	9	1
4	7	3
5	2	9
6	3	7
7	4	8
8	3	8

Мы можем расположить статьи на координатной плоскости в соответствии с количеством их упоминания в тексте, например, как показано на рис. 1.5. По горизонтали откладывается количество упоминаний первого ключевого слова, по вертикали – второго. Каждой статье соответствует точка на плоскости в соответствии с количеством упоминаний ключевых слов в ней.

Далее необходимо определить какую-либо структуру в скоплениях точек на плоскости, например, их можно выделить в два кластера, как показано в рис. 1.5. Первому кластеру могут соответствовать статьи политической тематики, а второму спортивной.

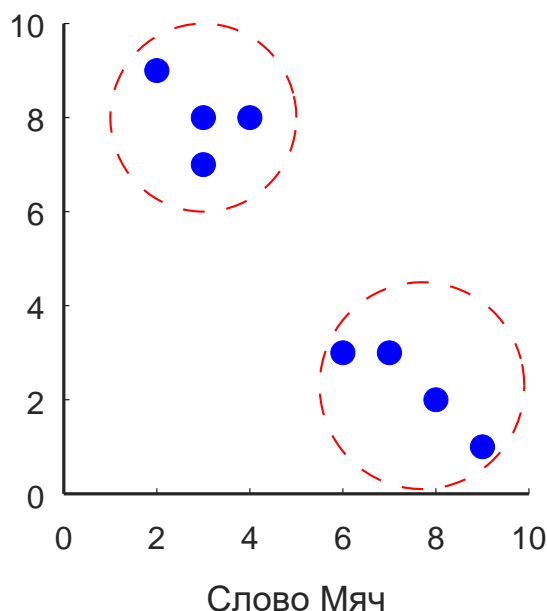


Рис. 1.5

Реальные системы информационного поиска имеют огромное число ключевых слов, которые могут автоматически пополняться, а также другие признаки статей, поэтому пространство выглядит намного сложнее, чем эта плоскость. ■

1.2 Задачи машинного обучения

Как уже было сказано, задачи машинного обучения разделяют на два больших класса: обучения с учителем (supervised learning) и обучения без учителя (unsupervised learning).

В задачах обучения с учителем исходные данные, на которых мы обучаем алгоритм, содержат правильные ответы (см. примеры 1 и 2 из предыдущего раздела). Основное отличие задач обучения без учителя (см. пример 3 из предыдущего раздела) является то, что правильных ответов в обучающих данных не содержится.

Задачу обучения с учителем определим в следующем виде.

Пусть $x^{(i)}$ – входные переменные, определенные в пространстве X (для примера 1 $x^{(i)}$ это площадь, X принадлежит R), $y^{(i)}$ – выходные переменные, которые необходимо предсказать, определенные в пространстве Y (в примере 1 $y^{(i)}$ это стоимость, Y принадлежит R). Пара $(x^{(i)}, y^{(i)})$ называется обучающим примером, а множество обучающих примеров $(x^{(i)}, y^{(i)})$, $i=1, \dots, t$ называется обучающей выборкой.

Задачей обучения с учителем является получение такой функции $h: X \rightarrow Y$, чтобы $h(x)$ являлось «хорошим» предсказанием для значения y . Функцию $h(x)$ называют гипотезой.

Пространство Y может быть определено произвольным образом.

Если $Y \in R$ (области действительных чисел), такие задачи называются задачами *регрессии*.

К примеру, задачей регрессии является задача предсказания стоимости дома по его параметрам.

Если $Y \in \{y_1, y_2, \dots, y_N\}$, $N \geq 2$ (некоторое ограниченное дискретное множество значений), такие задачи называются задачами *классификации*.

Например, задача медицинской диагностики, когда необходимо по ряду симптомов определить, является ли пациент здоровым или больным (в том числе, больным какой болезнью), относится к задачам классификации.

Мы рассмотрели несколько примеров задач машинного обучения из разных прикладных областей. Методы машинного обучения находят свое применение в огромном числе областей, вот лишь некоторые из них:

Медицинская диагностика

- Биоинформатика
- Техническая диагностика
- Финансовые приложения
- Информационный поиск
- Компьютерное зрение
- Распознавание речи и образов
- Обработка естественных языков

1.3. Функция стоимости

Рассмотрим задачу регрессии из примера 1. У нас имеется обучающая выборка из m примеров $(x^{(i)}, y^{(i)})$, $i=1, \dots, m$, которая представляет из себя пары (размер дома, стоимость дома) (см. табл. 4.).

Табл. 4

Размер, м ² – $x^{(i)}$	Стоимость, м.к. – $y^{(i)}$
2104	460
1416	232
1534	315
852	178
...	...

Реальная зависимость стоимости от размера (площади) является очень сложной, зависит от множества факторов и, самое главное, нам не известна.

Задачей регрессии является построение такой функции (гипотезы) $h(x)$, которая являлась бы «хорошим» предсказанием для значения y . Что это означает?

Пусть гипотеза будет линейной функцией:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

То есть мы будем предсказывать значения линейной функцией или линейной моделью. θ_0, θ_1 называют *параметрами модели*.

Разные значения параметров будут давать различный вид этой модели. На рис. 1.6, а показан функция $h_{\theta}(x) = 1.5$ (т.е. $\theta_0=1.5, \theta_1=0$), на рис. 1.6, б представлена функция $h_{\theta}(x)=0.5x$ (т.е. $\theta_0=0, \theta_1=0.5$), а на рис. 1.6, в – функция $h_{\theta}(x)=1 + 0.5x$ (т.е. $\theta_0=1, \theta_1=0.5$).

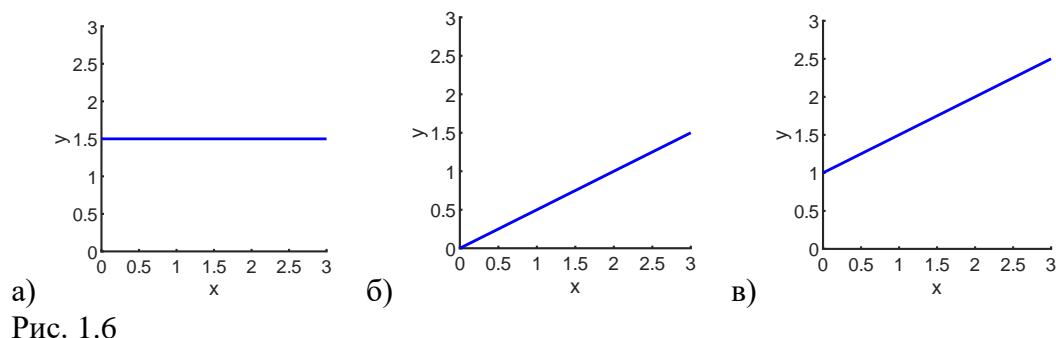


Рис. 1.6

Как выбрать значения параметров θ_0, θ_1 , которые устраивают нас лучше всего? Для начала необходимо решить, что означает «устраивают лучше всего».

Для задачи регрессии мы имеем обучающую выборку, которую можно отобразить на графике (рис. 1.7, точки обучающей выборки отображены красными маркерами).

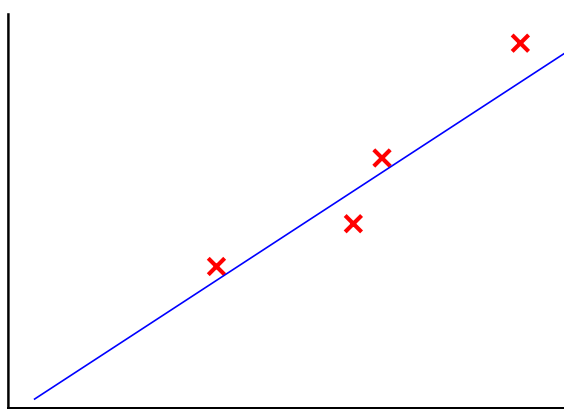


Рис. 1.7

Нам необходимо получить параметры θ_0, θ_1 такие, чтобы линия проходила через обучающую выборку наиболее близко ко всем точкам, например, как показано на рис. 1.7. Как понять, насколько близко линия проходит к точкам обучающей выборки?

Разницу между моделью и точками выборки определить просто: $q_i = h(x_i) - y_i$

Мы можем учесть все точки обучающей выборки: $Q = \sum_{i=1}^m (h(x_i) - y_i)$

Тогда, чем меньше значение Q , тем ближе гипотеза к реальным данным.

Обычно в машинном обучении применяют более сложную формулу:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x_i) - y_i)^2$$

Эта функция называется *функцией стоимости*. Математически она означает половину от среднего значения квадрата расстояния между значениями гипотезы и обучающей выборки. Поэтому еще такую функцию стоимости называют *функцией квадратичной ошибки*. Для большинства задач регрессии функция квадратичной ошибки является наилучшим вариантом, поэтому она широко применяется, хотя не является единственной функцией стоимости в задачах машинного обучения.

Наилучшей гипотезой для задачи регрессии является такая, которая имеет наименьшее значение функции стоимости. Целью линейной регрессии является нахождение таких параметров θ_0, θ_1 , чтобы минимизировать значение $J(\theta_0, \theta_1)$. В математике это называется *задачей оптимизации*.

Рассмотрим пример гипотезы $h(x) = \theta_0 + \theta_1 x$ при $\theta_0 = 0$. То есть функция $h(x)$ будет линейной, проходящий через центр координат под различными углами в зависимости от θ_1 .

Пусть обучающая выборка состоит из трех элементов: (1, 1), (2, 2) и (3, 3).

Каким будет значение функции стоимости $J(\theta_0, \theta_1)$ при различных значениях θ_1 ?

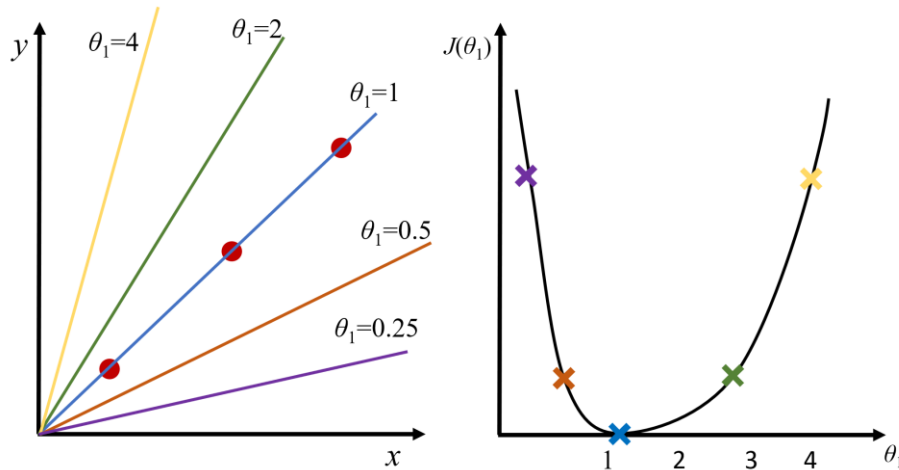


Рис. 1.8

Возьмем $\theta_1=1$. Легко вычислить:

$$J(\theta_0, \theta_1) = \frac{1}{2 \cdot 3} [(h(1) - 1)^2 + (h(2) - 2)^2 + (h(3) - 3)^2] = 0$$

При $\theta_1=1$ прямая проходит прямо через точки обучающей выборки и функция стоимости равна нулю.

Возьмем $\theta_1=0.5$, тогда

$$J(\theta_0, \theta_1) = \frac{1}{2 \cdot 3} [0.5^2 + 1^2 + 2^2] = 0.58$$

Можно построить графики функции $J(\theta_0, \theta_1)$ при различных значениях θ_1 (рис. 1.8).

Каждому значению θ_1 будет соответствовать отдельная прямая на левом графике, проходящая с некоторым наклоном через центр координат, на правом графике каждому значению θ_1 будет соответствовать точка значения функции стоимости для гипотезы с данным значением θ_1 . Для наглядности цвет линии на левом графике совпадает с цветом точки на правом графике.

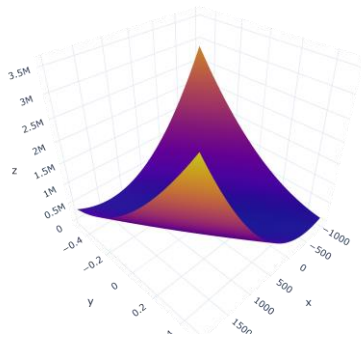
Задачей является определение такого параметра θ_1 , чтобы значение $J(\theta_0, \theta_1)$ было минимальным. Хорошо видно, что $J(\theta_0, \theta_1)$ будет минимальным при $\theta_1=1$. То есть $\theta_1=1$ является *глобальным минимумом* функции $J(\theta_0, \theta_1)$ для гипотезы $h(x)=\theta_0+\theta_1x$ при $\theta_0=0$.

Таким образом мы методом перебора различных значений θ_1 решили задачу оптимизации линейной функции. Конечно, метод перебора не является хорошим решением, и в машинном обучении есть более эффективные способы нахождения параметров θ , которые минимизируют функцию стоимости.

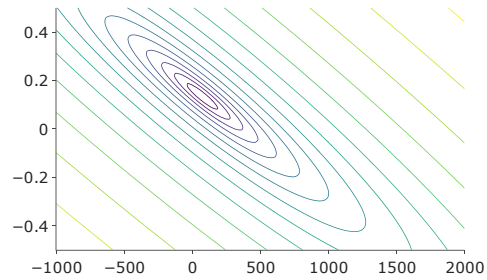
Рассмотрим еще один пример гипотезы $h(x)=\theta_0+\theta_1x$, в котором уже $\theta_0 \neq 0$, а будет также принимать различные значения. То есть теперь функция $J(\theta_0, \theta_1)$ является функцией от двух переменных.

Каждой паре значений (θ_0, θ_1) будет соответствовать некоторая прямая. Если вычислить для конкретной пары (θ_0, θ_1) значение функции стоимости $J(\theta_0, \theta_1)$, то в двумерном пространстве график $J(\theta_0, \theta_1)$ уже не так просто отобразить. Можно построить трехмерный график, показанный на рис. 1.9, а.

На графике точке в плоскости дна соответствует некоторое значение пары (θ_0, θ_1) , а высота трехмерной фигуры в данной точке является значением $J(\theta_0, \theta_1)$. У этой вогнутой фигуры также есть минимум, и значение (θ_0, θ_1) для этого минимума является наилучшим решением для линейной регрессии.



а)
Рис. 1.9



б)

Еще одним способом представления подобных функций от двух переменных является *контурный график* (рис. 1.9, б). В плоскости отображаются две оси для параметров θ_0, θ_1 . График функции $J(\theta_0, \theta_1)$ представляется собой набор концентрических контуров, в данном случае эллипсов, которые соответствуют точкам (θ_0, θ_1) с одинаковым значением $J(\theta_0, \theta_1)$. То есть все точки, лежащие на одном контуре, имеют одинаковые значения высоты в трехмерном графике. Такие контуры еще называют *линиями уровня*. Минимумом является значение в центре самой внутренней фигуры.

Рассмотрим примеры значений функции стоимости для различных гипотез. На графике (рис. 1.10) справа показана точка $\theta_0 = 800$, $\theta_1 = -0.15$ и соответствующий ей трафик гипотезы слева.

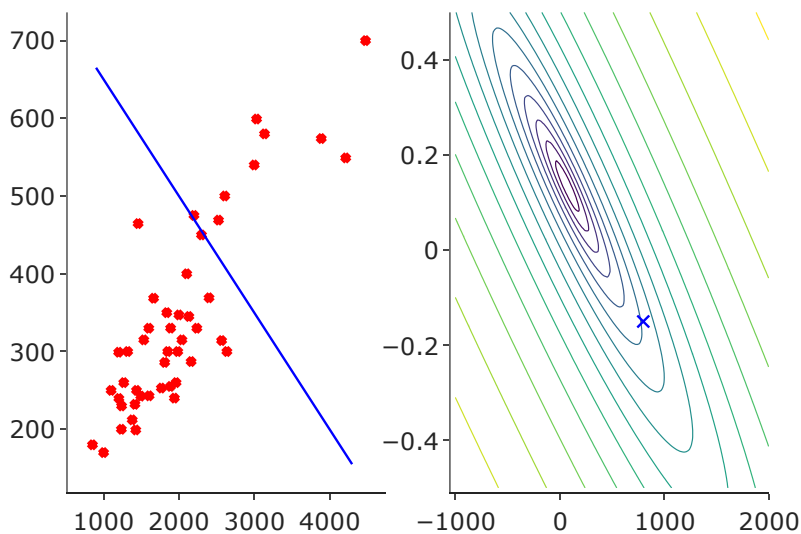


Рис. 1.10

Видно, что эта линия не очень хорошо проходит через точки обучающей выборки. Для точки $\theta_0 = 360$, $\theta_1 = 0$ значение $J(\theta_0, \theta_1)$ меньше, но такую регрессию все равно нельзя назвать хорошей (рис. 1.11).

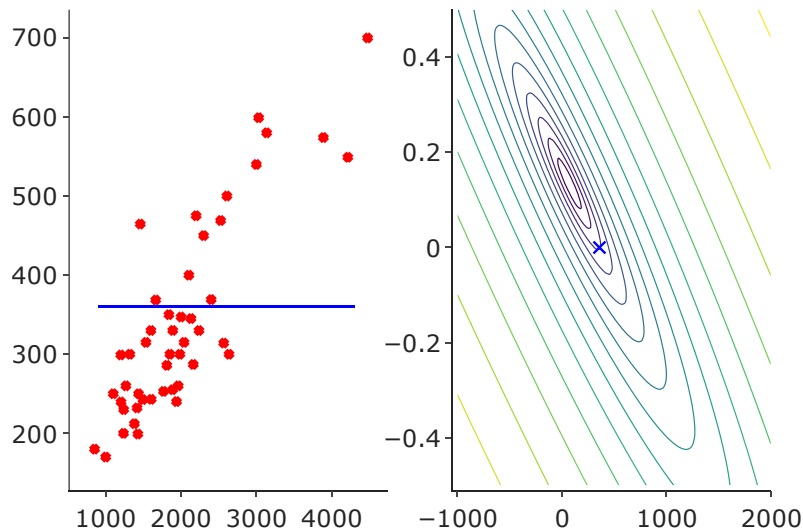


Рис. 1.11

Наконец, если взять точку $\theta_0 = 250$ $\theta_1 = 0.12$, которая находится близко к центру самого внутреннего концентрического эллипса, мы получим гипотезу, которая лучше всего из всех трех проходит через точки обучающей выборки (рис. 1.12).

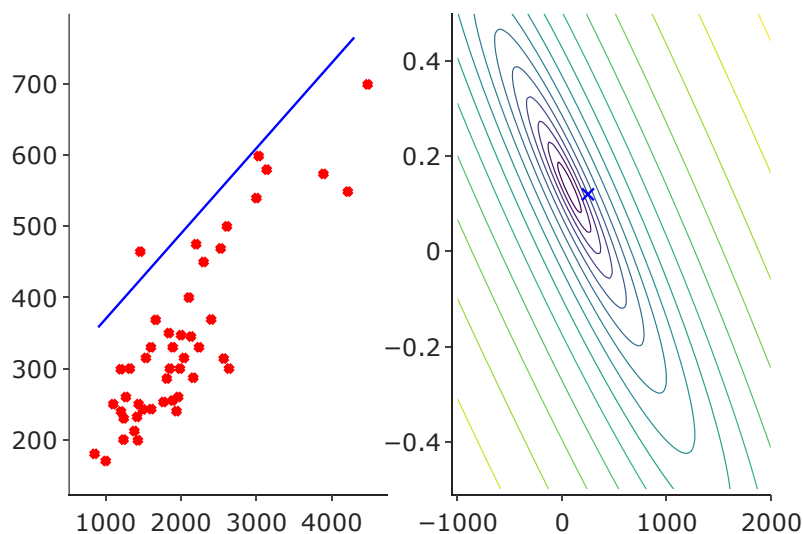


Рис. 1.12

Поскольку все точки обучающей выборки у нас не лежат на одной прямой, мы никогда не найдем такую прямую, которая будет проходить через них с нулевой функцией стоимости. Однако мы можем найти такую прямую, которая будет иметь самую минимальную стоимость, то есть быть ближе всего к общей совокупности точек обучающей выборки.

Конечно, перебором значений это делать очень долго и сложно. В машинном обучении есть методы, которые позволяют найти такие значения θ_0, θ_1 более эффективно. Эти методы называются *методами обучения*.

1.4. Обучение

Самым простым методом обучения является алгоритм *градиентного спуска*. Если дана функция $J(\theta_0, \theta_1, \dots, \theta_{n-1})$, алгоритм позволяет найти такие параметры $\theta_0, \theta_1, \dots, \theta_n$, при

которых достигается локальный минимум функции J :

$$\min_{\theta_0, \dots, \theta_n} J(\theta_0, \theta_1, \dots, \theta_n)$$

Для упрощения рассмотрим линейную регрессию $h(x) = \theta_0 + \theta_1 x$, когда $n=1$. Тогда функция стоимости зависит от двух параметров: $J(\theta_0, \theta_1)$.

Общий принцип алгоритма следующий:

1. Выбираем некоторые начальные значения θ_0, θ_1 . Например, можно выбрать $\theta_0=0, \theta_1=0$ или случайные значения.
2. Изменим θ_0, θ_1 на некоторое значение так, чтобы $J(\theta_0, \theta_1)$ уменьшилась.
3. Повторяем шаг 2 до тех пор, пока уменьшение $J(\theta_0, \theta_1)$ не станет достаточно малым, чтобы считать, что достигнут локальный минимум.

Предположим, график функции $J(\theta_0, \theta_1)$ выглядит, как показано на рис. 1.13.

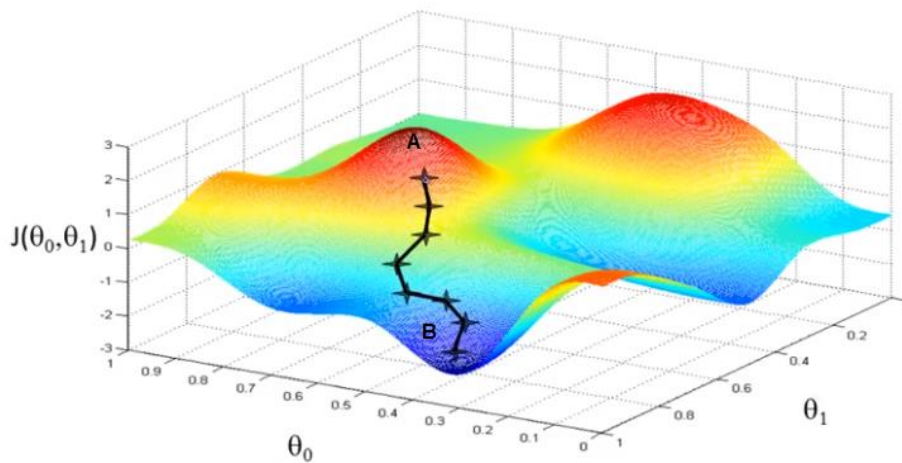


Рис. 1.13

Начнем с начальной точки A. Сделаем от этой точки небольшой шаг в сторону наибольшего склона функции J . Затем от новой точки сделаем еще такой шаг. В результате цепочки шагов, показанных на рисунке звездочками, мы окажемся в точке B, вокруг которой нет направлений для дальнейшего уменьшения. Точка B находится очень близко к локальному минимуму функции $J(\theta_0, \theta_1)$, то есть параметры θ_0, θ_1 в этой точке соответствуют одному из минимумов функции стоимости. Их можно воспринимать как решение задачи обучения линейной регрессии.

Можно провести аналогию работы алгоритма с катящимся с горы мячом. Он будет катиться в направлении самого крутого спуска, пока не достигнет ямы, в которой остановится.

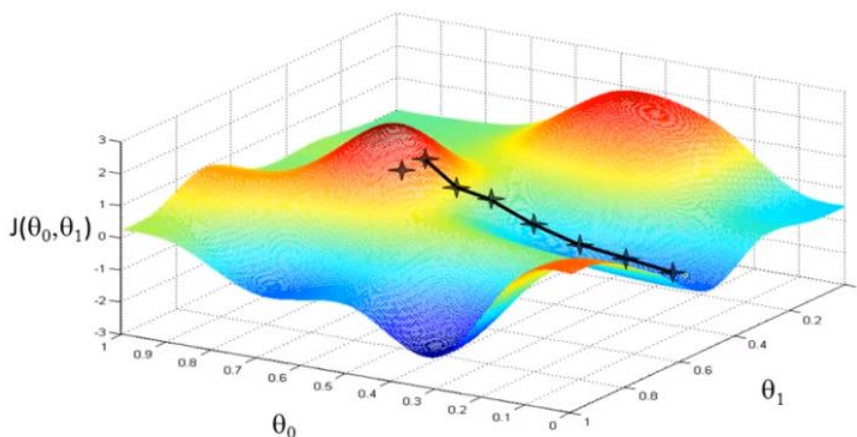


Рис. 1.14

Как видно на рисунке, точка минимума у функции стоимости может быть не одна. Если начать из другой начальной точки, то можно найти другие значения θ_0, θ_1 , для которых значение $J(\theta_0, \theta_1)$ является минимальным в некоторой области вокруг найденной точки (рис. 1.14). Поэтому эти минимумы называются локальными.

Самый минимальный из локальных минимумов является *глобальным минимумом*. В этой точке функция $J(\theta_0, \theta_1)$ принимает самое минимальное значение из всех возможных. Нахождение глобального минимума является наиболее желательным. Это самое лучшее решение задачи. Однако алгоритм градиентного спуска не дает гарантии его нахождения. Зачастую, если известно, что у функции стоимости больше одного локального минимума, алгоритм запускают несколько раз от различных начальных точек. Это также не дает гарантии нахождения глобального минимума, но повышает шансы.

Более строго алгоритм градиентного спуска можно определить так:

Алгоритм градиентного спуска.

Вход: $J(\theta_0, \dots, \theta_n)$ – функция, $\theta_0^0, \dots, \theta_n^0$ – начальные значения переменных, α – темп обучения, ε – критерий остановки.

Выход: $\theta_0^*, \dots, \theta_n^*$ – найденные значения для локального минимума.

Действия:

для $j = 0, \dots, n$

$$\theta_j := \theta_j^0$$

повторять {

для $j = 0, \dots, n$ {

$$d\theta_j := \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$$

}

для $j = 0, \dots, n$ {

$$\theta_j := \theta_j - \alpha d\theta_j$$

}

} пока $d\theta_j > \varepsilon, j = 0, \dots, n$

для $j = 0, \dots, n$

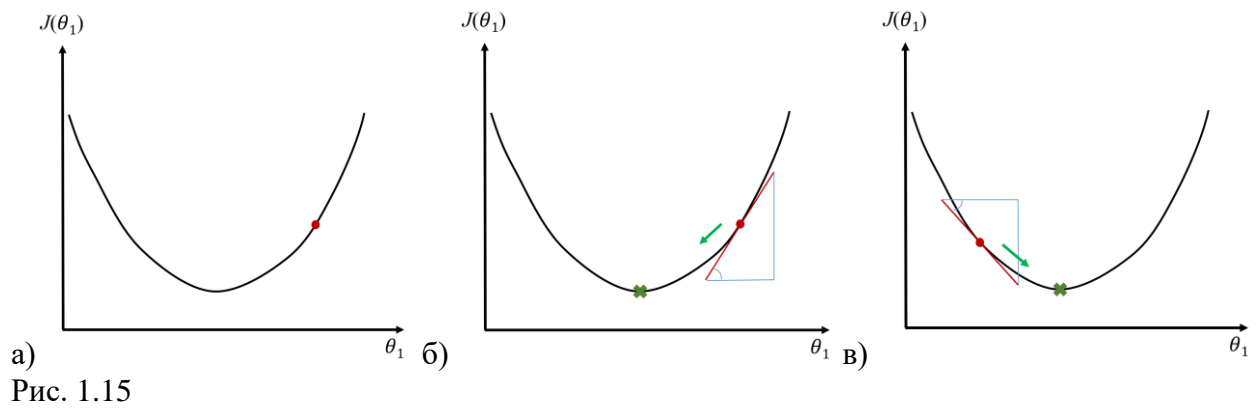
$$\theta_j^* := \theta_j$$

Конец алгоритма.

Параметр α называется *темпом обучения*. Он определяет шаг алгоритма. Если α будет достаточно большим, то алгоритм будет двигаться большими шагами. При малых α алгоритм сделает множество небольших шагов. Чем больше α , тем быстрее отработает алгоритм. Однако при больших α есть опасность «проскочить» точку локального минимума или «прыгать» вокруг нее, не попадая внутрь ямы. Подбор оптимального значения темпа обучения является отдельной задачей.

Часть $\frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$ называется *частной производной* функции $J(\theta_0, \dots, \theta_n)$ по переменной θ_j . В алгоритме мы вычисляем частные производные функции $J(\theta_0, \dots, \theta_n)$ по всем переменным $\theta_0, \dots, \theta_n$. Вектор из частных производных $\left(\frac{\partial}{\partial \theta_0} J(\theta_0, \dots, \theta_n), \dots, \frac{\partial}{\partial \theta_n} J(\theta_0, \dots, \theta_n) \right)$ называется *градиентом*. Этот вектор указывает на направление наибольшего возрастания функции $J(\theta_0, \dots, \theta_n)$. Двигаясь в противоположном градиенту направлении, можно прийти к минимуму. Поэтому этот алгоритм получил название градиентного спуска.

Для лучшего понимания, как выбирается направление спуска в алгоритме и как влияет темп обучения, рассмотрим работу алгоритма на функции одной переменной: $J(\theta_1), \theta_1 \in \mathbb{R}$. График функции представлен на рис. 1.15, а.



Предположим, что алгоритм находится в точке θ_1 . Тогда Новая точка, куда должен переместиться алгоритм вычисляется как:

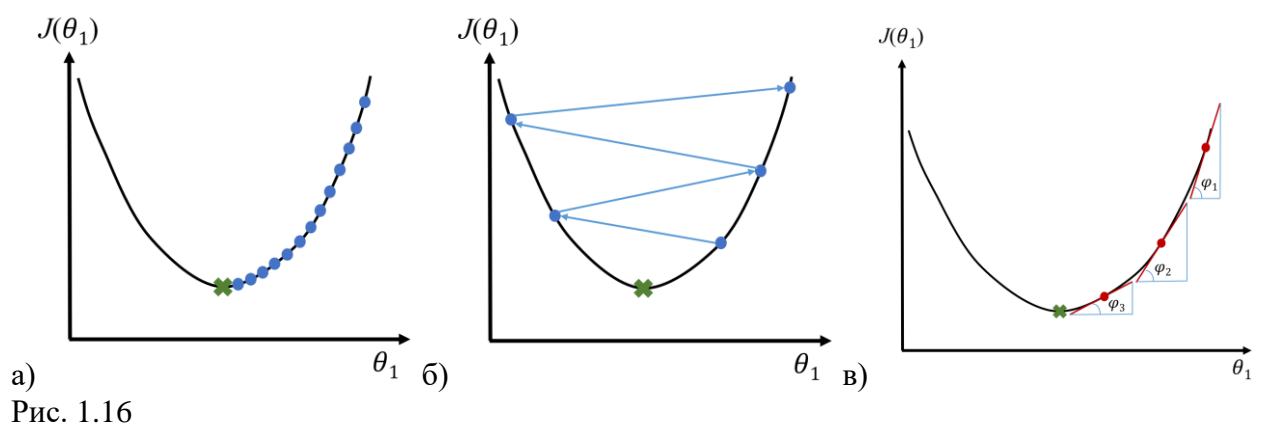
$$\theta_1 = \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

Частная производная в точке θ_1 фактически равна тангенсу угла касательной к функции $J(\theta_1)$ в точке θ_1 . В данной точке касательная имеет положительный угол. Значит значение частной производной тоже будет положительным (тангенс угла). Следовательно, от θ_1 будет отнято некоторое положительное значение и точка сместится влево, в сторону минимума функции (рис. 1.15, б).

Рассмотрим другой пример. Пусть начальная точка находится слева от минимума функции. В этом случае касательная будет проходить под отрицательным углом, производная будет иметь отрицательное значение. Отнимая по формуле отрицательное значение, мы сдвинем точку вправо, снова к минимуму функции (рис. 1.15, в).

Таким образом, благодаря производной (а в случае более чем одной переменной, вектору частных производных – градиенту), алгоритм движется в направлении локального минимума функции.

Рассмотрим теперь влияние темпа обучения на работу алгоритма. Если параметр α очень мал, то значение производной будет умножаться на малое значение, и шаг изменения текущей точки алгоритма будет небольшим. Как следствие, алгоритму придется сделать много шагов, чтобы достичь минимума (рис. 1.16, а).



Если же параметр α имеет очень большое значение, то шаг будет очень большой, и тогда алгоритм может перескочить через точку минимума. Это может привести к тому, что алгоритм не остановится (не сойдется), либо даже уйдет дальше от точки минимума, чем был в начальном состоянии (рис. 1.16, б).

Кроме того, величина шага алгоритма также зависит и от значения производной в текущей точке. Образно говоря, чем круче угол касательной в текущей точке, тем больше

его тангенс (по абсолютному значению), соответственно больше будет и величина шага алгоритма в положительную или отрицательную сторону.

Рассмотрим рис. 1.16, в. В первой точке склон кривой функции крутой, касательная к функции в этой точке будет проходить под большим углом к горизонтальной оси. Величина шага здесь больше. Переместившись во вторую точку, алгоритм вычисляет производную в ней. Угол касательной тут будет меньше, соответственно и шаг будет меньше. Расстояние от третьей точки до второй тоже меньше, чем было от второй до первой точки. Четвертая точка будет еще ближе к третьей, чем третья ко второй. Поскольку «крутизна» кривой становится все меньше при приближении к локальному минимуму, то длина шага все уменьшается. При этом значение параметра α имеет одинаковое значение. Длина шага уменьшается исключительно из-за уменьшения значения производной.

Таким образом, при приближении к минимуму алгоритм автоматически уменьшает длину шага, и нет необходимости изменять значение параметра α во времени, чтобы не пропустить минимум.

Рассмотрим теперь, как применяется алгоритм градиентного спуска в задаче линейной регрессии для нахождения оптимальных параметров линейной регрессионной функции на примере задачи предсказания стоимости дома по его площади (пример 2).

Задана гипотеза в виде линейной функции $h(x)=\theta_0+\theta_1x$. Функция стоимости задана в виде:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m ((h_{\theta}(x_i) - y_i)^2$$

Необходимо применить градиентный спуск и минимизировать функцию $J(\theta_0, \theta_1)$, получив значения θ_0, θ_1 , при которых гипотеза наиболее приближена к обучающей выборке.

Во-первых, необходимо найти частные производные $J(\theta_0, \theta_1)$ по каждому параметру θ_0, θ_1 :

$$\begin{aligned} \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) &= \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 = \frac{\partial}{\partial \theta_j} \frac{1}{2m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2, \quad j = 0, 1 \\ \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) &= \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) \\ \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) &= \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) x^{(i)} \end{aligned}$$

Если отобразить график функции стоимости $J(\theta_0, \theta_1)$ для линейной регрессии, то получится что-то подобное рис. 1.9. Особенностью этой функции является то, что она является выпуклой (или вогнутой). Она имеет только один минимум, он же является глобальным. Это означает, что решение, которое найдет алгоритм градиентного спуска, является самым лучшим решением, самым оптимальным.

С учетом найденных выше частных производных, алгоритм градиентного спуска для задачи линейной регрессии будет иметь вид:

Алгоритм градиентного спуска для линейной регрессии.

Вход: $J(\theta_0, \theta_1)$ – функция, θ_0^0, θ_1^0 – начальные значения переменных, α – темп обучения, ε – критерий остановки.

Выход: θ_0^*, θ_1^* – найденные значения для локального минимума.

Действия:

$$\theta_0 := \theta_0^0$$

$$\theta_1 := \theta_1^0$$

повторять {

$$\begin{aligned}
d\theta_0 &:= \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) \\
d\theta_1 &:= \frac{1}{m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) x^{(i)} \\
\theta_0 &:= \theta_0 - \alpha d\theta_0 \\
\theta_1 &:= \theta_1 - \alpha d\theta_1 \\
\} \text{ пока } d\theta_0 > \varepsilon, d\theta_1 > \varepsilon \\
\theta_0^* &:= \theta_0 \\
\theta_1^* &:= \theta_1
\end{aligned}$$

Конец алгоритма.

Обратите внимание, что вычисление $d\theta_0$ и $d\theta_1$ выполняются перед изменением значений θ_0 и θ_1 . Если поместить вычисление $d\theta_0$ и изменение θ_0 в одно выражение, то мы получим ошибку: вычисление $d\theta_1$ будет выполняться уже с другим значением θ_0 . Поэтому градиент сначала вычисляется по обоим координатам, а затем уже применяется для шага алгоритма.

Рассмотрим, как будет работать алгоритм градиентного спуска по шагам. На рис. 1.17 в левой части отображена обучающая выборка и нарисована линия гипотезы $h(x) = \theta_0 + \theta_1 x$, соответствующая тем значениям параметров θ_0, θ_1 , которые получены на текущем шаге алгоритма градиентного спуска. Возьмем начальную точку $\theta_0 = 900, \theta_1 = -0.1$ (рис. 1.17, а).

На рис. 1.17, б показан один из промежуточных шагов. В левой части отображена прямая гипотезы при текущих значениях $\theta_0 = 400, \theta_1 = -0.02$. В правой части рисунка отображается контурный график функции стоимости от переменных θ_0 (по горизонтальной оси), θ_1 (по вертикальной оси). Синими маркерами отображаются точки значений θ_0, θ_1 всех пройденных шагов алгоритма.

На последнем графике алгоритм градиентного спуска приходит в глобальный минимум, и этому минимуму соответствует функция гипотезы, показанная на левом графике. Она хорошо «ложится» на множество обучающей выборки, имеет наименьшую функцию стоимости (то есть минимальную среднюю квадратичную ошибку от всех точек обучающей выборки). Эту функцию гипотезы можно использовать для предсказания неизвестных значений, которые не встречаются в обучающей выборке. Например, что стоимость дома площадью 3000 будет составлять 485000 (зеленый маркер на рис. 1.17, в).

Таким образом, мы решили задачу линейной регрессии для предсказания стоимости дома по его площади на основе обучающей выборки из примера 2.

Рассмотренный алгоритм еще называют *обычным градиентным спуском* или *полным градиентным спуском*, поскольку в нем на каждом шаге используются все значения обучающей выборки.

Существуют другие версии алгоритма градиентного спуска, которые на каждом шаге рассматривают не все множество обучающей выборки, а некоторое ее подмножество. Есть более продвинутые алгоритмы обучения, которые позволяют получить решение за меньшее число шагов и найти глобальный минимум функции стоимости. Эти алгоритмы рассматриваются в разделе 3.7.

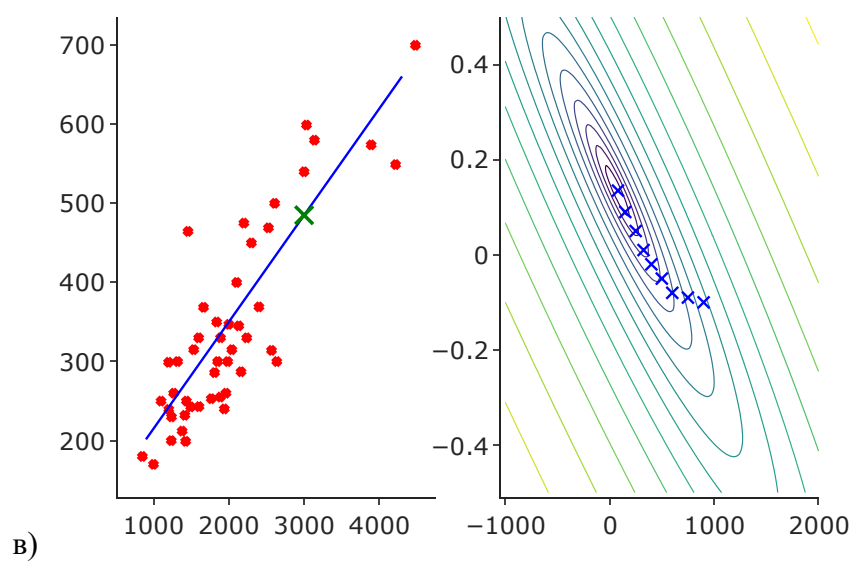
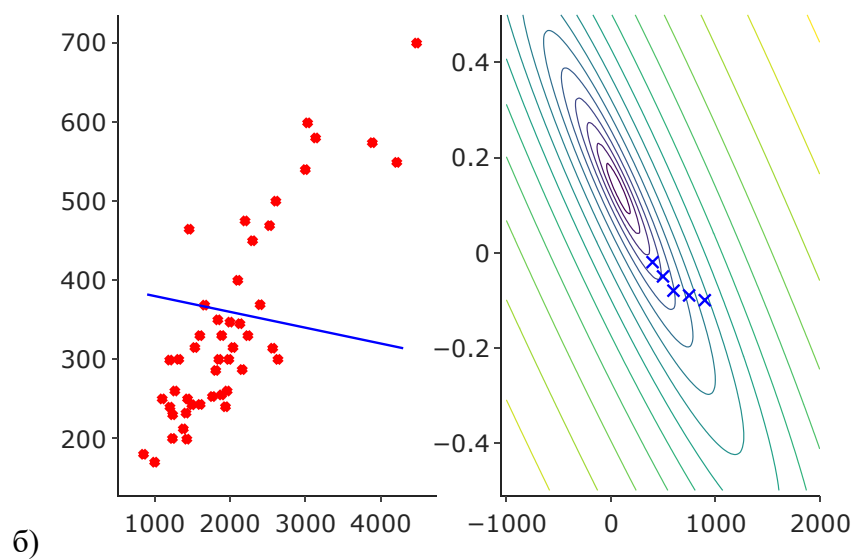
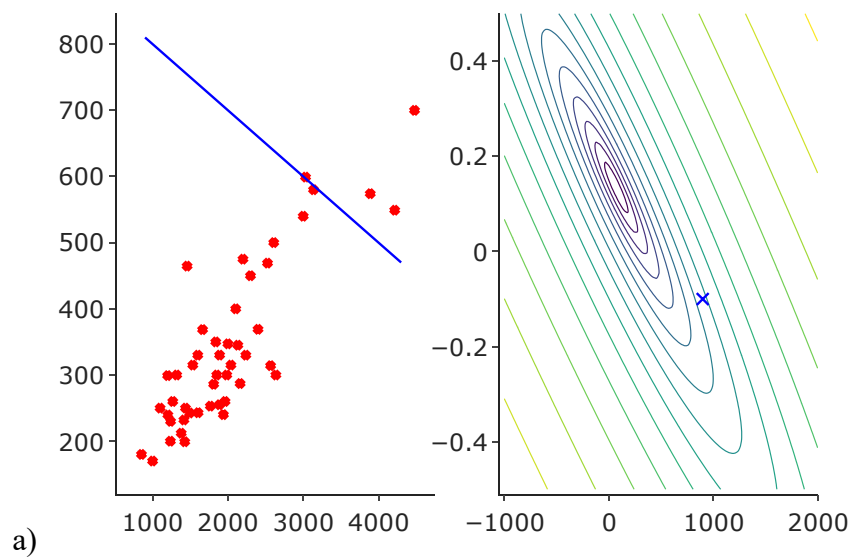


Рис. 1.17

1.5. Основы линейной алгебры

Матрицей называют двумерный массив следующего вида:

$$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$$

Число строк m и столбцов n матрицы называют *размерами матрицы* и записывают $m \times n$. Матрица выше является матрицей 3×3 . Когда хотят обозначить множество всех возможных матриц заданного размера, то пишут $\mathbb{R}^{m \times n}$.

Пример матрицы 3×2 :

$$\begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 147 & 1437 \end{bmatrix}$$

Для обращения к элементам матрицы пишут A_{ij} , что означает элемент матрицы A , находящийся в ячейке на пересечении строки i и столбца j .

Для матрицы:

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 147 & 1437 \end{bmatrix}$$

элемент $A_{11} = 1402$, $A_{12} = 191$, $A_{21} = 1371$, $A_{32} = 1437$. Элементы с индексами за пределами размеров матрицы являются неопределенными, обращение к ним ошибка: $A_{23} = \text{undefined}$.

Вектором является матрица размерами $n \times 1$, то есть с одним столбцом и n строками:

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

Число n называют *длиной вектора* или *размерностью вектора*. Множество векторов размерностью n , состоящих из действительных чисел, обозначают \mathbb{R}^n .

Элемент вектора обозначают y_i , где i – номер строки элемента. Например, $y_1 = 460$, $y_3 = 315$. При нумерации элементов вектора можно придерживаться одного из двух подходов, *индексации на базе нуля*, или *индексации на базе единицы*.

Индексация на базе нуля начинает отсчет самого верхнего элемента вектора с 0, а самый последний имеет индекс $n-1$:

$$y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

Индексация на базе единицы начинает отсчет самого верхнего элемента вектора с 1, а самый последний имеет индекс n :

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Выбор способа индексации зависит от предпочтений или удобства. Многие алгоритмы записываются компактнее и проще, если использовать 0-индексацию.

Над векторами возможны операции: сложения, умножения, транспонирования, обращения.

При сложении матриц размером $m \times n$ получается матрица размером $m \times n$, элементы которой являются суммой соответствующих элементов слагаемых матриц. Например:

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 4 & 0.5 \\ 2 & 5 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1+4 & 0+0.5 \\ 2+2 & 5+5 \\ 3+0 & 1+1 \end{bmatrix} = \begin{bmatrix} 5 & 0.5 \\ 4 & 10 \\ 3 & 2 \end{bmatrix}$$

Сложение матриц разного размера запрещено и является ошибкой:

$$\begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} + \begin{bmatrix} 5 & 9 \\ 1 & 0.7 \end{bmatrix} = error$$

Операция сложения матриц является *коммутативной*, то есть допускает перестановку мест правого и левого слагаемого: $A + B = B + A$.

Результатом операции умножения матрицы на число является матрица той же размерности, каждый элемент которой умножен на число. Например:

$$3 \times \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 9 \end{bmatrix} = \begin{bmatrix} 3 \times 1 & 3 \times 0 \\ 3 \times 2 & 3 \times 5 \\ 3 \times 3 & 3 \times 9 \end{bmatrix} = \begin{bmatrix} 3 & 0 \\ 6 & 15 \\ 9 & 27 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 9 \end{bmatrix} \times 3$$

Операция умножения матрицы на число является коммутативной, как и сложение.

Можно также разделить матрицу на число (если это число не ноль):

$$\begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} / 4 = \frac{1}{4} \times \begin{bmatrix} 4 & 0 \\ 6 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1.5 & 3/4 \end{bmatrix}$$

Можно комбинировать несколько операций над матрицами в одном выражении:

$$3 \times \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix} - \begin{bmatrix} 2 \\ 4 \\ 8 \end{bmatrix} / 2 = \begin{bmatrix} 3 \\ 12 \\ 6 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 5 \end{bmatrix} - \begin{bmatrix} 1 \\ 2 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 11 \\ 7 \end{bmatrix}$$

Приоритеты операций над матрицами аналогичны приоритетам над простыми числами: умножение и деление выполняются перед сложением и вычитанием.

Умножение матрицы на вектор осуществляется по правилу: $A^{m \times n} \times x^n = y^m$. То есть, при умножении матрицы $m \times n$ на вектор $n \times 1$ получается вектор $m \times 1$. При этом элементы вектора результат рассчитываются как сумма произведений каждой строки матрицы на столбец вектора:

$$y_i = \sum_{j=1}^n A_{ij} \cdot x_j, \quad i = 1, \dots, m$$

Например:

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 3 \cdot 5 \\ 4 \cdot 1 + 0 \cdot 5 \\ 2 \cdot 1 + 1 \cdot 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 5 \end{bmatrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

Еще один пример:

$$\begin{bmatrix} 1 & 2 & 1 & 5 \\ 0 & 3 & 0 & 4 \\ -3 & -2 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 3 + 1 \cdot 2 + 5 \cdot 1 \\ 0 \cdot 1 + 3 \cdot 3 + 0 \cdot 2 + 4 \cdot 1 \\ -1 \cdot 1 + (-2) \cdot 3 + 0 \cdot 2 + 0 \cdot 1 \end{bmatrix} = \begin{bmatrix} 14 \\ 13 \\ -7 \end{bmatrix}$$

Операция умножения матрицы на матрицу очень похожа на умножение матрицы на

вектор. Мы можем умножить матрицу размером $m \times n$ на матрицу размером $n \times o$ и получить матрицу размером $m \times o$:

$$A^{m \times n} \times B^{n \times o} = C^{m \times o}$$

Значения элементов матрицы C при этом вычисляются следующим образом:

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj}, \quad i = 1, \dots, m, \quad j = 1, \dots, o.$$

Пример:

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 3 \cdot 0 + 2 \cdot 5 & 1 \cdot 3 + 3 \cdot 1 + 2 \cdot 2 \\ 4 \cdot 1 + 0 \cdot 0 + 1 \cdot 5 & 4 \cdot 3 + 0 \cdot 1 + 1 \cdot 2 \end{bmatrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} = \begin{bmatrix} 11 \\ 9 \end{bmatrix} \quad \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} = \begin{bmatrix} 11 \\ 9 \end{bmatrix} \quad \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

В отличие от операции сложения матриц, умножение матрицы на матрицу не является коммутативной операцией, то есть

$$A \times B \neq B \times A.$$

Пример с матрицами 2×2 :

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \end{bmatrix}$$

Если перемножаемые матрицы имеют разные размеры:

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \\ 2 & 1 & 0 \\ 3 & 5 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} = \begin{bmatrix} 11 & 10 \\ 9 & 14 \\ 2 & 7 \\ 8 & 16 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 0 & 1 \\ 5 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \\ 2 & 1 & 0 \\ 3 & 5 & 1 \end{bmatrix} = error$$

При умножении матрицы на матрицу, количество строк второй матрицы должно совпадать с количеством столбцов первой матрицы, при этом количество строк первой матрицы и столбцов второй матрицы могут быть различны, и они определяют размеры получаемой в результате умножения матрицы. Если мы поменяем местами перемножаемые матрицы, то количество столбцов первой может уже не совпадать с количеством строк второй.

При этом операция умножения матрицы на матрицу является *ассоциативной*, то есть можно перемножать матрицы в различных сочетаниях:

$$A \times (B \times C) = (A \times B) \times C.$$

Например:

$$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} = ?$$

$$\text{Вариант 1: } \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}; \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 0 & 0 \end{bmatrix}$$

$$\text{Вариант 2: } \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \times \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 4 & 2 \end{bmatrix}; \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} 4 & 2 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 0 & 0 \end{bmatrix}$$

Единичная матрица является такой матрицей, умножение которой на любую другую матрицу дает в результате вторую матрицу без изменений, то есть:

$$I \times A = A \times I = A$$

Среди действительных чисел есть число 1, при умножении на которое любое число остается неизменным: $1 \cdot z = z \cdot 1 = z$. Единичная матрица является матричным аналогом числа 1. Она представляет собой квадратную матрицу с диагональными единицами, остальные значения равны нулю. Единичную матрицу заданного размера $m \times m$ обозначают $I^{m \times m}$. Примеры единичных матриц:

$$I^{2 \times 2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$I^{3 \times 3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$I^{1 \times 1} = [1]$$

Для упрощения написания больших единичных матриц пропускают повторение нулей и единиц, заменяя серию одинаковых цифр троеточием, при необходимости добавляя в правом верхнем углу размер матрицы:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix}^{4 \times 4}$$

При умножении на единичную матрицу необходимо выбирать размер единичной матрицы так, чтобы он подходил под размер умножаемой матрицы (количество столбцов первой матрицы-множителя должно совпадать с количеством строк второй матрицы-множителя). Пример:

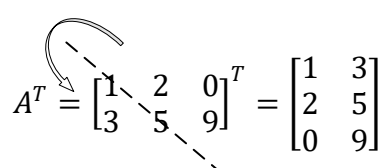
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \times \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 2 \cdot 0 & 1 \cdot 0 + 2 \cdot 1 \\ 3 \cdot 1 + 4 \cdot 0 & 3 \cdot 0 + 4 \cdot 1 \\ 5 \cdot 1 + 6 \cdot 0 & 5 \cdot 0 + 6 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & \dots & 0 \\ \vdots & 1 & \vdots \\ 0 & \dots & 1 \end{bmatrix}^{3 \times 3} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 + 0 \cdot 3 + 0 \cdot 5 & 1 \cdot 2 + 0 \cdot 4 + 0 \cdot 6 \\ 0 \cdot 1 + 1 \cdot 3 + 0 \cdot 5 & 0 \cdot 2 + 1 \cdot 4 + 0 \cdot 6 \\ 0 \cdot 1 + 0 \cdot 3 + 1 \cdot 5 & 0 \cdot 2 + 0 \cdot 4 + 1 \cdot 6 \end{bmatrix} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Над матрицей возможна операция транспонирования, в результате которой индексы столбцов и строк элементов матрицы меняются местами. Операция транспонирования обозначается большой буквой T в верхнем индексе обозначения матрицы: A^T .

Сама матрица при этом зеркально отображается относительно диагональной линии из верхнего левого угла:

$$A = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}$$


После транспонирования матрица размером $m \times n$ превращается в матрицу $n \times m$.

Еще одной операцией над матрицами является *инверсия* или получение *обратной матрицы*. Она обозначается верхним правым индексом -1 . Для действительного числа инверсия возвращает другое действительное число, при умножении на которое получается единица: $z \cdot z^{-1} = 1$. Например:

$$3^{-1} = \frac{1}{3}$$

$$3 \cdot 3^{-1} = 3 \cdot \frac{1}{3} = 1$$

Не все числа могут иметь обратное число. Например, число 0 не имеет обратного числа, поскольку на ноль делить нельзя.

По аналогии с действительными числами, обратная матрица – это такая матрица, при умножении на которую в результате получается единичная матрица:

$$A \times A^{-1} = A^{-1} \times A = I$$

Инверсия применима только к квадратным матрицам $m \times m$, у которых число столбцов и строк совпадает.

Пример:

$$A = \begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix}$$

$$AA^{-1} = \begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix} \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Не все матрицы могут иметь обратную, например, матрица

$$\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

не имеет обратной. Такие матрицы называются *вырожденными*.

Способ получения обратной матрицы не прост. Существует несколько целых алгоритмов для ее вычисления. Но мы их рассматривать не будем.

Линейная алгебра широко применяется в машинном обучении. Возьмем пример с задачей предсказания стоимости квартиры по ее площади, рассмотренный ранее.

Пусть имеется линейная гипотеза $h_\theta(x) = -40 + 0.25x$. Возьмем значения площадей квартир из некоторого набора данных. Например, это будут числа: 2104, 1416, 1534, 852. Запишем их в виде матрицы, в которой два столбца – первый всегда равен 1, а второй значениям площадей квартир:

$$X = \begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix}$$

Параметры гипотезы запишем в виде вектора:

$$H = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} = \begin{bmatrix} -40 \\ 0.25 \end{bmatrix}$$

Теперь мы можем получить значения предсказанных по гипотезе стоимостей квартир за одну матричную операцию умножения:

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix} \times \begin{bmatrix} -40 \\ 0.25 \end{bmatrix} = \begin{bmatrix} 486 \\ 314 \\ 344 \\ 173 \end{bmatrix}$$

Таким образом мы получили за одно матричное умножение сразу четыре предсказанных значения. Этим значений может быть и больше, если дополнит матрицу X новыми строками.

Можно пойти еще дальше, и за одну операцию выполнить несколько предсказаний от разных гипотез. Пусть имеется три гипотезы:

$$h_1(x) = -40 + 0.25x$$

$$h_2(x) = 200 + 0.1x$$

$$h_3(x) = -150 + 0.4x$$

Сформируем из значений параметров θ_0, θ_1 этих гипотез матрицу, в которой каждой гипотезе соответствует столбец, в котором в первой строке записана θ_0 , во второй θ_1 :

$$H = \begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix}$$

Теперь мы можем получить матрицу предсказанных значений:

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix} \times \begin{bmatrix} -40 & 200 & -150 \\ 0.25 & 0.1 & 0.4 \end{bmatrix} = \begin{bmatrix} 486 & 410 & 692 \\ 314 & 342 & 416 \\ 344 & 353 & 464 \\ 173 & 285 & 191 \end{bmatrix}$$

$\leftarrow x_1$
 $\leftarrow x_2$
 $\leftarrow x_3$
 $\leftarrow x_4$

\uparrow \uparrow \uparrow
 $h_1(x)$ $h_2(x)$ $h_3(x)$

Каждому столбцу матрицы соответствуют значения предсказания для всех искомым данных, полученных по одной гипотезе. Можно добавить любое количество исходных данных и любое количество гипотез. Это увеличивает размер матриц, но все равно вычисление их всех выполняется за одну операцию матричного умножения.

В современных языках программирования имеется множество библиотек, которые позволяют выполнить матричные операции очень быстрым и эффективным способом. Многие из них для ускорения расчета задействуют функции распараллеливания на многоядерных и многопроцессорных компьютерах, используют для расчета возможности графических видеокарт, которые по своей сути являются очень мощными матричными процессорами и способны выполнять миллиарды матричных умножений в секунду. Это позволяет выполнять вычисления предсказаний в машинном обучении во много раз быстрее, чем программировать вычисление значений традиционным способом через циклы.

Вопросы и задания

1. Существует программа, которая изучает, какие электронные письма были отмечены пользователем как спам, а какие как не спам. Основываясь на этих данных, программа определяет в дальнейшем, является ли новое полученное письмо спамом или не является. Основываясь на определении Митчелла, ответьте, что является задачей Т, опытом Е и критерием Р для этой программы? (классификация писем на спам/не спам; база данных меток пользователя; количество писем, правильно классифицированных как спам).

2. Определите, к какому классу задач (регрессии или классификации) относятся следующие проблемы:

а) у вас есть склад товаров. Вам нужно решить, сколько товаров будет продано в следующем месяце?

б) вам нужно написать программу, которая оценит учетные записи почтовых ящиков пользователей и решит, какие из них могут быть взломаны в будущем?

в) в кредитную организацию обращается клиент за займом на покупку автомобиля. Необходимо решить, выдавать ему кредит или нет?

3. Какие из следующих задач относятся к обучению с учителем, а какие без?

а) имеется база e-mail писем, для которых указано, которые из них являются спамом. Задача научить алгоритм определять спам.

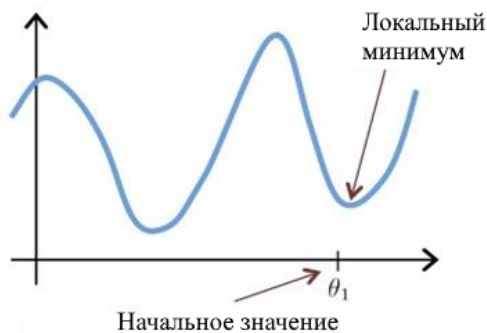
б) имеется база новостных заметок, которые необходимо разделить на группы по близким темам.

в) дана база медицинских данных пациентов, больных диабетом. Необходимо научиться определять, имеет ли пациент диабет.

г) дана база данных о потребителях (список товаров, которые потребитель покупал), необходимо автоматически определить наиболее популярные сегменты товаров и их покупателей

д) имеется база данных переходов посетителей веб-сайтов (с одного сайта на другой), необходимо сгруппировать сайты на группы, внутри которых выполняется большая часть переходов.

4. Предположим, что начальная точка θ_1 находится в локальном минимуме функции $J(\theta_1)$, как показано на рисунке:



Чему будет равно значение θ_1 после выполнения шага алгоритма градиентного спуска?

Варианты:

а) не изменится

б) изменится в случайном направлении

в) изменится в направлении глобального минимума функции

г) уменьшится

д) увеличится

5. Какие из следующих утверждений верны?

а) чтобы обеспечить сходжение алгоритма градиентного спуска, необходимо медленно уменьшать значение параметра α с каждым шагом;

б) градиентный спуск гарантирует нахождение глобального минимума функции;

в) градиентный спуск может сойтись при фиксированном значении α (однако при слишком малых или слишком больших значениях, алгоритм может не сойтись);

г) для функции средней квадратичной ошибки, применяемой в качестве функции стоимости в линейной регрессии, нет локальных минимумов, отличных от глобального.

2. МНОЖЕСТВЕННАЯ ЛИНЕЙНАЯ РЕГРЕССИЯ

2.1 Множественная линейная регрессия

Ранее мы предсказывали стоимость дома в зависимости от его размера, то есть строили линейную регрессионную модель от одной переменной:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Теперь добавим с эту модель еще немного данных: размер дома, количество комнат, этажей, возраст дома. Таким образом набор данных будет у нас выглядеть следующим образом:

Размер, м ²	Комнат	Этажей	Возраст дома	Цена
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...

Эти параметры называют *переменными* или *признаками* и обозначают обычно x_i , где i – номер признака (колонки данных). В данном случае: x_1 – площадь дома, x_2 – количество комнат, x_3 – количество этажей, x_4 – возраст дома. Стоимость дома – переменную, которую мы собираемся предсказывать, обозначим y .

Введем также обозначения:

n – количество признаков элемента обучающей выборки (в нашем примере $n=4$);

$x^{(i)}$ – вектор признаков i -го элемента обучающей выборки,

$$x^{(i)} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

например

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$$

$x_j^{(i)}$ – j -й признак вектора признаков i -го элемента обучающей выборки, например, $x_3^{(2)} = 2$.

Поскольку $n=4$, то $x^{(i)} \in \mathbb{R}^4$. m – размер обучающей выборки, $m=47$ (в таблице приведены только 4 первые строки).

Гипотеза линейной регрессии от нескольких переменных имеет вид:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$$

или в нашем примере:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Конкретный вид гипотезы имеет какие-то значения параметров $\theta_0, \dots, \theta_n$, например

$$h_{\theta}(x) = 80 + 0.1x_1 + 15x_2 - 0.5x_3 - 2x_4$$

Данную гипотезу можно расшифровать так: каждый дом имеет базовую стоимость 80, которая увеличивается в зависимости от площади (с коэффициентом 0.1), увеличивается от числа комнат (с коэффициентом 15), уменьшается с количеством этажей (по 0.5 за этаж) и уменьшается с возрастом дома (с коэффициентом 2).

Линейная гипотеза может быть записана более компактно в матричной форме. Введем новый параметр $x_0 = 1$. Он не входит в набор данных, это константа, которую мы вво-

дим для облегчения дальнейших записей. С его учетом в каждом элементе обучающей выборки добавляется значение $x_0^{(i)} = 1$, и с учетом этого вектор признаков из вектора \mathbb{R}^n становится вектором \mathbb{R}^{n+1} :

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \in \mathbb{R}^{n+1}$$

Индексация в векторе начинается с нуля.

Приведенную выше гипотезу мы можем теперь записать как:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Мы также можем записать параметры $\theta_0, \dots, \theta_n$ в виде вектора размером $n + 1$:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} \in \mathbb{R}^{n+1}$$

Теперь можно записать гипотезу как произведение этих двух векторов:

$$h_\theta(x) = \theta^T x$$

$$h_\theta(x) = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix}^T \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = [\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3 \quad \theta_4] \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$$

Таким образом задается множественная линейная регрессия в общем виде. Чтобы решить задачу регрессии, то есть найти такую функцию $h_\theta(x)$, которая будет наиболее правильно предсказывать значения y , необходимо найти параметры $\theta_0, \dots, \theta_n$. Для этого также может применяться метод градиентного спуска.

Функция стоимости задается аналогично тому, как она была задана для случая предсказания от одного параметра:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

только необходимо учесть, что θ и $x^{(i)}$ являются векторами размерности $n + 1$. Напомним, что m – количество элементов в обучающей выборке, т.е. $i = 1, \dots, m$.

В данном случае $J(\theta)$ является функцией от $n + 1$ переменных. Для определения шага градиентного спуска, необходимо вычислить частные производные функции стоимости по каждому из параметров: $\frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$, $j = 0, \dots, n$. В результате получим следующее:

$$\begin{aligned} \frac{\partial}{\partial \theta_0} J(\theta) &= \frac{1}{m} \sum_{i=0}^m (h_\theta(x^{(i)}) - y^{(i)}) \\ \frac{\partial}{\partial \theta_1} J(\theta) &= \frac{1}{m} \sum_{i=0}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)} \\ &\dots \\ \frac{\partial}{\partial \theta_n} J(\theta) &= \frac{1}{m} \sum_{i=0}^m (h_\theta(x^{(i)}) - y^{(i)}) x_n^{(i)} \end{aligned}$$

Принимая во внимание введенное ранее обозначение $x_0^{(i)} = 1$, можно записать в более общем и компактном виде:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=0}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}, \quad j = 0, \dots, n$$

Используя это выражение можно реализовать описанный ранее алгоритм градиентного спуска для множественной линейной регрессии.

Алгоритм градиентного спуска для множественной регрессии.

Вход: $J(\theta_0, \dots, \theta_n)$ – функция, $\theta_0^0, \dots, \theta_n^0$ – начальные значения переменных, α – темп обучения, ε – критерий остановки.

Выход: $\theta_0^*, \dots, \theta_n^*$ – найденные значения для локального минимума.

Действия:

для $j = 0, \dots, n$:

$$\theta_j := \theta_j^0$$

повторять {

для $j = 0, \dots, n$:

$$d\theta_j := \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x^{(i)}$$

для $j = 0, \dots, n$:

$$\theta_j := \theta_j - \alpha d\theta_j$$

} пока $d\theta_j > \varepsilon, j = 0, \dots, n$

для $j = 0, \dots, n$:

$$\theta_j^* := \theta_j$$

Конец алгоритма.

Критерием остановки алгоритма в случае множества переменных является одно из условий:

- 1) Шаг градиента по одной из координат меньше заданной константы (именно этот вариант показан в алгоритме выше)
- 2) Расстояние между соседними точками меньше заданной константы
- 3) Расстояние между соседними значениями функции меньше заданной константы
- 4) Длина градиента меньше заданной константы

2.2. Нормализация данных

Рассмотрим пару признаков из примера с предсказанием стоимости квартиры: x_1 – площадь и x_2 – количество комнат. Значения x_1 лежат в пределах от 0 до 3000, тогда как значения x_2 в пределах от 1 до 5 в обучающей выборке.

Если нарисовать контурный график функции стоимости от этих двух параметров, то получим рис. 2.1.

Эллипсы будут очень вытянутыми. Запустив алгоритм градиентного спуска на таких данных, мы увидим, что алгоритм перемещается по нему зигзагами. Это происходит из-за большой разницы масштабов между параметром x_1 и x_2 . Там, где по параметру x_1 алгоритм уверенно идет в сторону уменьшения функции стоимости, по параметру x_2 изменения очень малы и больше похоже на случайные скачки. Если критерий останова алгоритма будет довольно мал, алгоритм может совсем не остановиться, потому что хаотичные прыжки по параметру x_2 будут больше значения критерия останова.

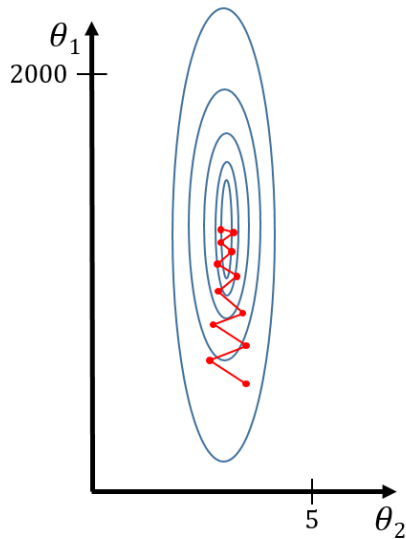


Рис. 2.1

Выходом из этой ситуации является *масштабирование* исходных данных. Разделим значение каждого параметра на его максимальное значение:

$$x_1 = \frac{\text{площадь}}{3000} \quad x_2 = \frac{\text{кол-во комнат}}{5}$$

Теперь максимальное значение каждой переменной не превышает единицы, и они примерно равны друг другу. График функции стоимости от этих двух параметров будет выглядеть, как показано на рис. 2.2, и градиентный спуск по таким данным будет работать быстро и точно.

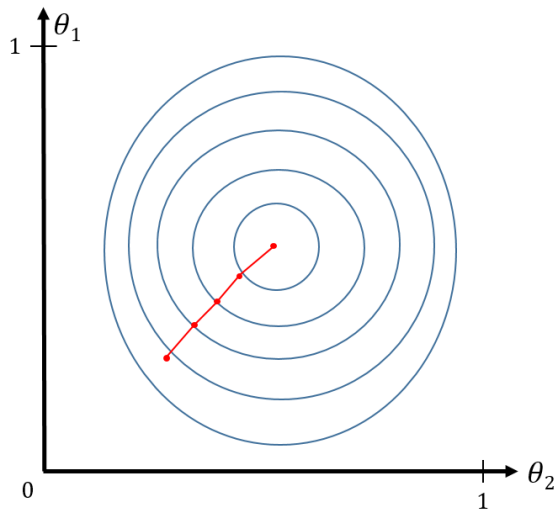


Рис. 2.2

Еще одним полезным приемом при подготовке исходных данных является *нормализация среднего значения*.

Если средним значением площади квартиры является 1500, а количество комнат в среднем 2, то нормализованные значения вычисляются как:

$$x_1 = \frac{\text{площадь} - 1500}{3000} \quad x_2 = \frac{\text{кол-во комнат} - 2}{5}$$

Общая формула нормализации данных:

$$\hat{x}_i = \frac{x_i - \mu_i}{s_i},$$

где μ_i – среднее значение переменной (математическое ожидание), $s_i = \max(x_i) - \min(x_i)$ – ширина диапазона значения переменной (часто вместо s_i используют среднеквадратическое отклонение).

В этом случае все значения переменных попадают в диапазон $-0.5 \leq x_i \leq 0.5$, а средним значением переменной является 0 (рис. 2.3).

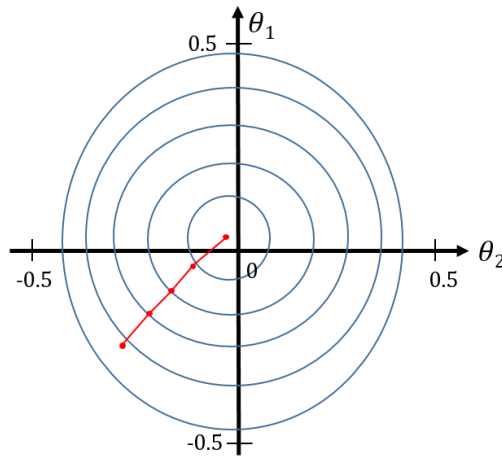


Рис. 2.3

В общем случае, рекомендуется масштабировать исходные данные так, чтобы их значения попали приблизительно в диапазон $-1 \leq x_i \leq 1$. Эти границы являются условными. Так, например, значения $0 \leq x_i \leq 3$ или $-2 \leq x_i \leq 0.9$ являются приемлемыми. Однако границы $-10 \leq x_i \leq 200$ или $-0.001 \leq x_i \leq 0.00001$ необходимо масштабировать.

Помимо нормализации данных, еще одним важным пунктом эффективного применения алгоритма градиентного спуска является правильный выбор параметра темпа обучения α . Как уже обсуждалось ранее, слишком малый параметр приводит к долгому времени сходимости алгоритма. Слишком большое значение может привести к тому, что алгоритм не сойдется и не закончит работу.

Есть один способ проверить корректность работы алгоритма и выбор параметра темпа обучения. Для этого необходимо отобразить график значения функции стоимости от текущих параметров θ на каждом шаге алгоритма (рис. 2.4).

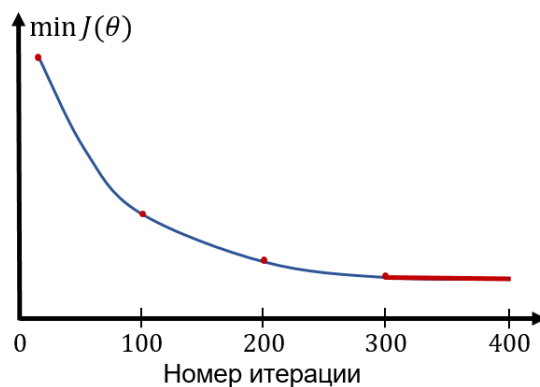


Рис. 2.4

По горизонтальной оси откладывается номер итерации алгоритма, а по вертикальной – значение $J(\theta)$. При правильной работе алгоритма значение $J(\theta)$ должно уменьшаться на каждом шаге. Как правило, на первых шагах значение $J(\theta)$ уменьшается более интенсивно, затем скорость его снижения уменьшается. Чем больше значение α , тем круче снижается график $J(\theta)$ на первых шагах. После некоторого номера итерации алгоритма значение $J(\theta)$

уменьшается практически незаметно, например, после 300 итерации на графике. Дальнейшая работа алгоритма становится уже напрасной. Поэтому одним из критериев остановки алгоритма является очень малое (меньше заданного значения) уменьшение функции $J(\theta)$.

Так выглядит «правильная» работа алгоритма. Однако, можно получить и другие варианты графика.

Например, график может выглядеть так, как показано на рис. 2.5.

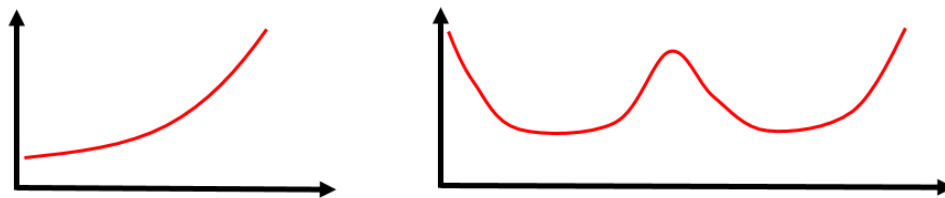


Рис. 2.5

Это означает, что выбрано очень большое значение для параметра темпа обучения. В этом случае, из-за большого шага алгоритм промахивается мимо точки минимума. Он может подходить ближе к минимуму, а потом опять отдаляться от него, но не подходит к минимуму (говорят, алгоритм *не сходится*).

При очень малом значении алгоритм не промахнется мимо минимума, но может возникнуть ситуация, что он будет очень долго очень маленькими шагами к нему приближаться, затрачивая минуты и часы на решение задачи, которая при правильном выборе шага решалась бы за секунды или доли секунды.

Таким образом, выбор правильного значения темпа обучения является достаточно важной задачей. Эффективных универсальных методов расчета «правильного» значения не существует. Простейшим практическим способом является подбор значения α в некотором диапазоне и проверка поведения алгоритма. Например, пробовать значения 0.0001, 0.001, 0.01, 0.1 и т.п. с тем, чтобы найти приемлемые границы для темпа обучения. Окончательное значение темпа обучения рекомендуется выбирать немного меньше, чем наибольшее из «правильных» значений. Это обеспечивает хорошую скорость сходимости.

2.3. Выбор и конструирование переменных

На данный момент мы знаем, как работает линейная регрессия от нескольких переменных. В качестве переменной для функции гипотезы мы выбирали один из признаков элемента обучающей выборки. Однако, мы можем взять не все признаки, взять некоторый признаки более одного раза и даже сконструировать новые переменные из нескольких признаков, и получить новые более мощные решения.

Возьмем задачу предсказания стоимости дачного участка, который описывается всего двумя признаками: длиной участка вдоль улицы (frontage) и глубиной от улицы (depth), как на рис. 2.6.



Рис. 2.6

Мы можем построить регрессионную модель вида:

$$h_{\theta}(x) = \theta_0 + \theta_1 \cdot \text{frontage} + \theta_2 \cdot \text{depth}$$

Но мы можем сконструировать из этих двух признаков новую переменную:

$$x = \text{frontage} \cdot \text{depth}$$

и получить построенную регрессионную модель всего от одного признака – площади участка:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

Конструируя собственные переменные, мы можем получать более качественные модели, и даже применять аппарат линейной регрессии с градиентным спуском для построения более сложных нелинейных гипотез.

Пусть имеется задача предсказания стоимости дома от его размера. Обучающая выборка для этой модели выглядит следующим образом (рис. 2.7)

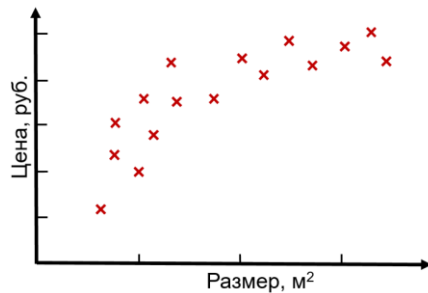


Рис. 2.7

Хорошо видно, что точки выборки не лежат на одной прямой, обычная линейная регрессия вряд ли даст нам хорошее предсказание стоимости дома. Распределение этих точек больше похоже на квадратичную функцию:

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

которая может выглядеть следующим образом (рис. 2.8, а).

Возможно, что эта функция не лучшее решение. Ведь квадратичная функция с возрастанием пойдет вниз (рис. 2.8, б).

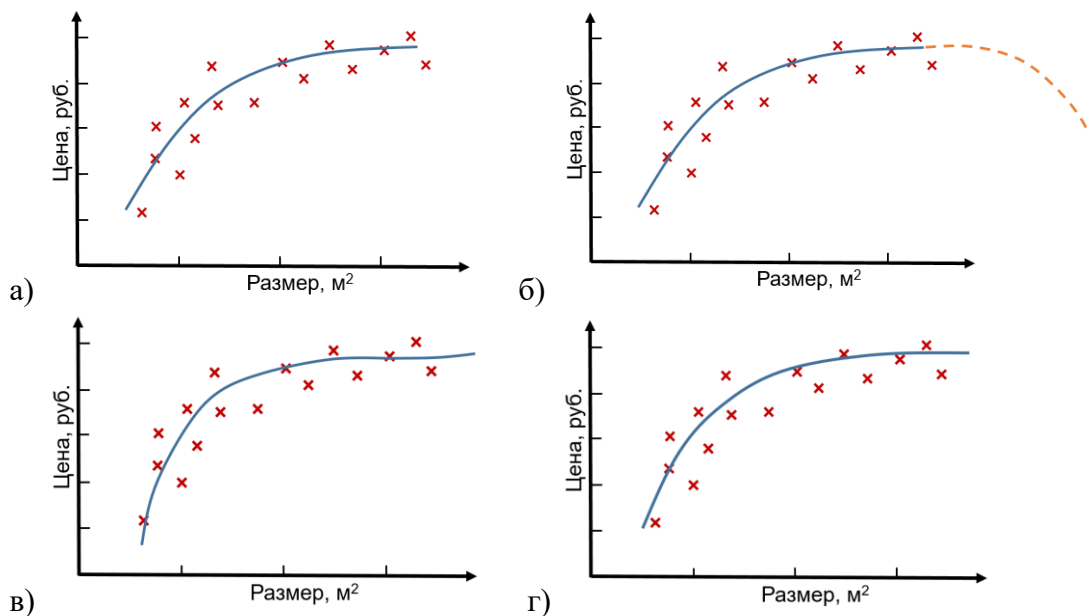


Рис. 2.8

Вряд ли можно считать правильной модель, которая полагает, что стоимость дома будет уменьшаться при увеличении его размера. Можно выбрать другую полиномиальную модель, например, кубическую функцию (рис. 2.8, в):

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

Хотя эта функция нелинейная, сделать это можно с помощью аппарата линейной регрессии. Возьмем гипотезу от трех признаков:

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

и зададим признаки следующего вида:

$$x_1 = (size)$$

$$x_2 = (size)^2$$

$$x_3 = (size)^3$$

Вычислив x_1, x_2, x_3 можно применить алгоритм градиентного спуска и получить оптимальные значения параметров θ . Стоит учесть, что, если мы используем степенные функции, необходимо обратить особое внимание на масштабирование параметров. Если $size$ варьируется в диапазоне 1-1000, тогда для x_1 диапазон значений до 1000, для x_2 он вырастает до 1 000 000, а для x_3 уже до 10^9 . Так что масштабирование становится особенно важным.

Переменные могут быть конструированы и другими способами, например, можно взять функцию квадратного корня:

$$h_{\theta}(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

И получить еще одну достаточно адекватную по сравнению с квадратичной модель (рис. 2.8, г).

Эта модель, в отличие от квадратичной, не уменьшает стоимость при увеличении размера, как и кубическая. Но мы этого добились с использованием только двух переменных, а не трех.

2.4. Нормальные уравнения

Метод нормальных уравнений позволяет во многих случаях получить решение задачи регрессии быстрее и проще, чем алгоритм градиентного спуска. Если градиентный спуск подразумевает пошаговое приближение к точке минимума, то подход с использованием нормальных уравнений дает возможность найти оптимальные значения параметров θ аналитически за один шаг. Конечно, нормальные уравнения имеют и недостатки, о которых мы поговорим позже.

Суть подхода с использованием нормальных уравнения заключается в следующем.

Гипотеза линейной регрессии нескольких переменных в общем случае выглядит следующим образом:

$$h_{\theta}(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Функция стоимости линейной регрессии:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

В точке минимума функции стоимости ее производная равна нулю:

$$\frac{\partial}{\partial \theta} J(\theta) = 0$$

Поскольку θ является вектором размерности $n + 1$ ($\theta \in \mathbb{R}^{n+1}$), чтобы определить аналитически значения параметров θ , при которых $J(\theta)$ минимальная, необходимо решить систему $n + 1$ дифференциальных уравнений:

$$\begin{cases} \frac{\partial}{\partial \theta_0} J(\theta_0, \dots, \theta_n) = 0 \\ \frac{\partial}{\partial \theta_1} J(\theta_0, \dots, \theta_n) = 0 \\ \dots \\ \frac{\partial}{\partial \theta_n} J(\theta_0, \dots, \theta_n) = 0 \end{cases}$$

Мы не будем решать эту систему, а рассмотрим сразу результат ее решения. Переменные в исходных данных мы можем представить в виде матрицы $X \in \mathbb{R}^{m \times n+1}$, параметры $\theta \in \mathbb{R}^{n+1}$, а $y \in \mathbb{R}^m$. Решение представленной выше системы дифференциальных уравнений очень компактно записывается в матричной форме:

$$\theta = (X^T X)^{-1} X^T y$$

Пользуясь этим выражением, можно легко найти оптимальные значения θ .

Пример. Имеем задачу предсказания цен на недвижимость со следующей обучающей выборкой ($n=4$, $m=47$):

Размер, м ²	Комнат	Этажей	Возраст дома	Цена
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

Для ее решения будет использована гипотеза:

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Построим матрицу параметров обучающей выборки и вектор решений:

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \quad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \\ \dots \end{bmatrix}$$

Здесь $X \in \mathbb{R}^{47 \times 5}$, $\theta \in \mathbb{R}^{5 \times 1}$, $y \in \mathbb{R}^{47 \times 1}$.

Вектор оптимальных параметров гипотезы вычисляется через нормальное уравнение:

$$\theta = (X^T X)^{-1} X^T y = \left(\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}^T \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix} \right)^{-1} \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \\ \dots & \dots & \dots & \dots & \dots \end{bmatrix}^T \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \\ \dots \end{bmatrix}$$

■

В большинстве случаев нормальное уравнение позволяет получить результат быстрее и проще, чем алгоритм градиентного спуска. Более того, нормальные уравнения нечувствительны к разнице масштабов исходных переменных, а значит, для них нет необходимости выполнять нормализацию данных, как для градиентного спуска.

Однако есть ограничения, связанные с тем, что в нормальном уравнении необходимо вычислить обратную матрицу $(X^T X)^{-1}$.

Поскольку $X \in \mathbb{R}^{m \times n+1}$, то $X^T X \in \mathbb{R}^{(n+1) \times (n+1)}$.

Нахождение обратной матрицы имеет вычислительную сложность порядка $O(n^3)$.

Для небольших значений n ($n < 1000$) нормальные уравнения имеют существенное преимущество перед градиентным спуском. При $n \gg 1000$ это преимущество падает. Для $n > 10^6$ решение через нормальное уравнение может быть очень долгим.

Вторая сложность связана с тем, что не все матрицы являются обратимыми. Если $X^T X$ окажется необратимой, то вычислить θ через нормальное уравнение не получится.

В ряде случаев необратимость $X^T X$ является следствием неподходящих исходных данных.

Во-первых, если в исходных данных имеются две (или более) переменных, линейно зависимых друг от друга, то $X^T X$ может оказаться необратимой. Например, если у нас имеется переменная x_1 (размер в метрах), и в этих же данных имеется переменная x_2 (размер в сантиметрах). Очевидно, что они линейно зависимы: $x_1 = 100 \cdot x_2$. Если $X^T X$ оказывается необратимой, необходимо проверить данные на наличие линейно зависимых переменных и исключить лишние переменные.

Во-вторых, если размер обучающей выборки существенно меньше количества переменных (например, $n=50$ и $m=15$), то матрица $X^T X$ с высокой долей вероятности окажется необратимой. В этом случае необходимо добиться, чтобы $m \geq n$. Например, увеличить размер обучающей выборки или, если это невозможно, убрать из выборки некоторые малозначительные переменные.

Если ни один из этих способов не помогает, то можно прибегнуть к процедуре *регуляризации переменных*, о которой речь пойдет позже.

Вопросы и задания

1) Задана функция стоимости для множественной линейной регрессии в виде:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)})^2$$

Какие из выражений ниже являются эквивалентными ей?

а) $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (\theta^T x^{(i)} - y^{(i)})^2$

б) $J(\theta) = \frac{1}{2m} \sum_{i=1}^m \left(\left(\sum_{j=0}^n \theta_j x_j^{(i)} \right) - y^{(i)} \right)^2$

в) $J(\theta) = \frac{1}{2m} \sum_{i=1}^m \left(\left(\sum_{j=1}^n \theta_j x_j^{(i)} \right) - y^{(i)} \right)^2$

г) $J(\theta) = \frac{1}{2m} \sum_{i=1}^m \left(\left(\sum_{j=0}^n \theta_j x_j^{(i)} \right) - \left(\sum_{j=0}^n y_j^{(i)} \right) \right)^2$

2) Используется алгоритм обучения для задачи предсказания стоимости дома. Одна из переменных в обучающей выборке означает возраст дома и принимает значения от 30 до 50 со средним значением 38. Какое из следующих выражений следует использовать для нормализации этой переменной?

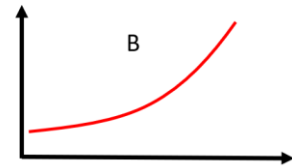
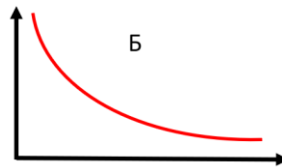
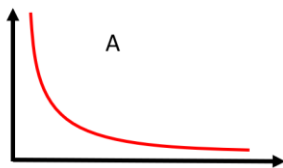
а) $x_i = \text{возраст дома}$

б) $x_i = \frac{\text{возраст дома}}{50}$

в) $x_i = \frac{\text{возраст дома} - 38}{50}$

г) $x_i = \frac{\text{возраст дома} - 38}{20}$

3) Предположим, что алгоритм градиентного спуска был запущен три раза со значениями $\alpha=0.01$, $\alpha=0.1$ и $\alpha=1$. Определите, какому из графиков ниже (А, Б и В) соответствует какое из значений α .



а) А для $\alpha=0.01$, Б для $\alpha=0.1$, В для $\alpha=1$

б) А для $\alpha=0.1$, Б для $\alpha=0.01$, В для $\alpha=1$

в) А для $\alpha=1$, Б для $\alpha=0.01$, В для $\alpha=0.1$

г) А для $\alpha=1$, Б для $\alpha=0.1$, В для $\alpha=0.01$

4) Имеем обучающую выборку из $m=23$ промеров с $n=5$ переменными. Решаем задачу регрессии с использованием нормального уравнения $\theta = (X^T X)^{-1} X^T y$. Какие будут размерности матриц и векторов θ , X , y ?

5) Имеем обучающую выборку из $m=50$ промеров с $n=15$ переменными. Необходимо решить задачу линейной регрессии от нескольких переменных. Что лучше выбрать: алгоритм градиентного спуска или нормальные уравнения и почему?

3. ЛОГИСТИЧЕСКАЯ РЕГРЕССИЯ

3.1. Задача классификации

В разделах 1 и 2 мы подробно разобрали, как при помощи линейной функции можно решать задачу регрессии. Рассмотрим теперь задачу классификации. Как показано в разделе 1.2, задача классификации предполагает, что ответы обучающей выборки принадлежат ограниченному множеству значений: $y \in \{y_1, y_2, \dots, y_N\}$, $N \geq 2$. Примерами задач классификации выступают определение писем электронной почты (спам/не спам), медицинская диагностика (болен/здоров) и т.д.

Самой простой является задача классификации на два класса ($N=2$). Рассмотрим, например, задачу диагностики ракового заболевания (доброкачественный/злокачественный). Обозначим $y=0$ – диагностируемая опухоль доброкачественная (опасности не представляет), $y=1$ – злокачественная (требуется лечения).

Имеется обучающая выборка из значений x – размер опухоли, y – доброкачественная/злокачественная.

Мы можем попробовать использовать уже известный нам аппарат линейной регрессии и попытаться решить задачу предсказания диагностики заболевания по размеру опухоли. Отобразим обучающую выборку на графике (рис. 3.1).

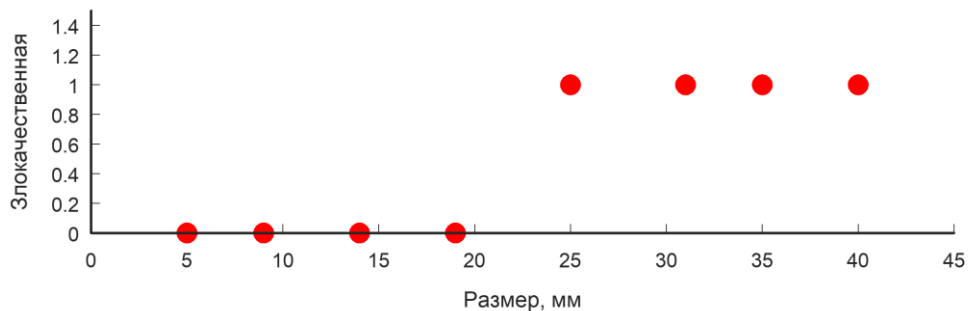


Рис. 3.1

По данной выборке можно построить линейную регрессионную модель вида $h_\theta(x) = \theta^T x$, обучить ее методом градиентного спуска и получить прямую линию, которая будет проходить, например, как показано на рис. 3.2 (зеленая прямая).

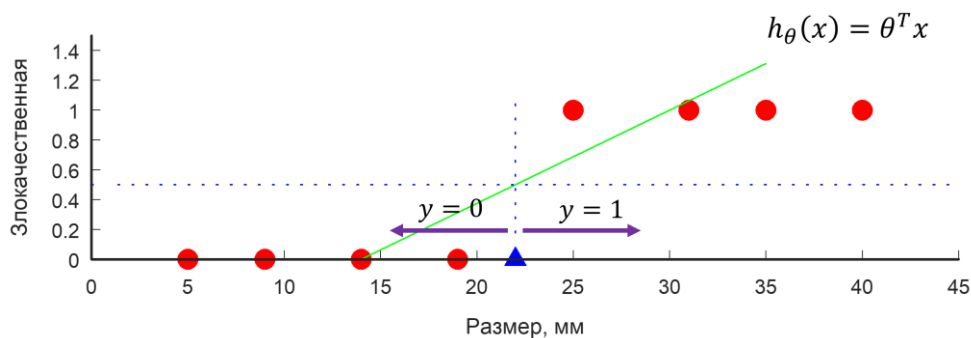


Рис. 3.2

Хорошо видно, что если $h_\theta(x) \geq 0.5$, то опухоль является злокачественной ($y=1$ в обучающей выборке), а если $h_\theta(x) < 0.5$, то опухоль является доброкачественной ($y=0$). То есть, добавив к линейной регрессии пару условий можно использовать ее для решения задачи классификации.

Однако, имеются определенные подводные камни в таком подходе, затрудняющие использование линейной регрессии для классификации в чистом виде.

Предположим, что обучающая выборка имеет еще несколько точек с большими значениями x , как показано на рис. 3.3.

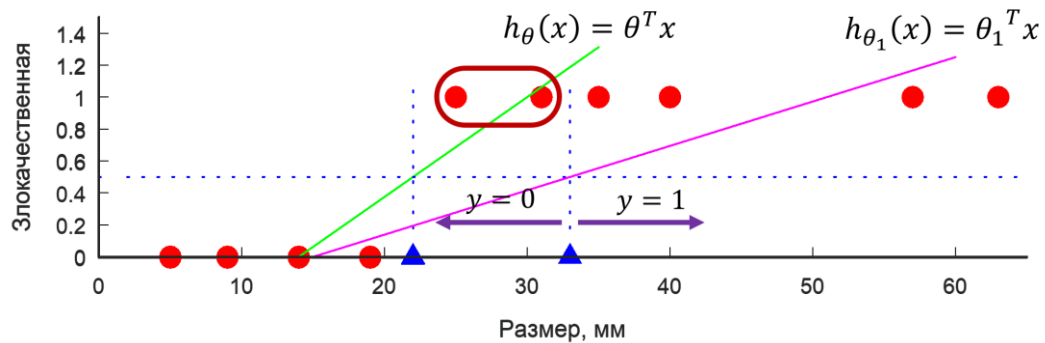


Рис. 3.3

В таком случае мы получим другую регрессионную модель $h_{\theta_1}(x) = \theta_1^T x$, и прежние условия ($y=1: h_{\theta}(x) \geq 0.5$; $y=0: h_{\theta}(x) < 0.5$) перестают работать. На рис. 3.3 красной линией обведены две точки, которые будут ошибочно классифицироваться, как доброкачественные.

Тем не менее, с небольшими изменениями этот подход можно использовать для классификации. В первую очередь, эти изменения касаются вида функции гипотезы.

3.2. Гипотеза логистической регрессии

Функция гипотезы для линейной регрессии в матричном виде представляет собой следующее выражение:

$$h_{\theta}(x) = \theta^T x,$$

где θ и x – являются векторами. Значение $h_{\theta}(x) \in \mathbb{R}$, то есть может принимать любое значение.

Для задачи классификации на два класса нам было бы удобно использовать такую функцию гипотезы, значения которой были бы ограничены диапазоном $[0, 1]$.

Такая функция имеется, это логистическая функция (или сигмовидная функция, или сигмоида):

$$g(z) = \frac{1}{1 + e^{-z}}.$$

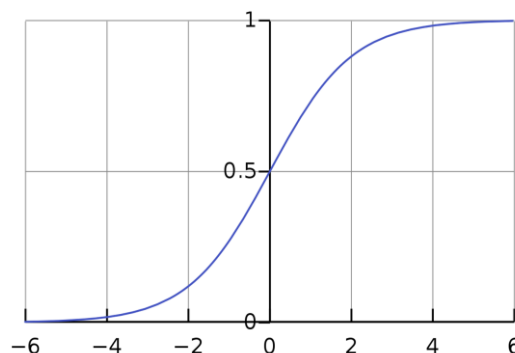


Рис. 3.4

График этой функции приведен на рис. 3.4. В нуле функция проходит через 0.5, на бесконечности стремится к 1, а на минус бесконечности стремится к нулю.

Для задач классификации используют регрессионную модель с гипотезой вида:

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}.$$

То есть линейная регрессия дополнительно обрамляется логистической функцией. Такая модель называется *логистической регрессией*.

При этом значение функции $h_{\theta}(x) \in [0,1]$ интерпретируется как вероятность того, что для заданного x класс $y=1$:

$$h_{\theta}(x) = P(y = 1|x; \theta).$$

Например, $h_{\theta}(x) = 0.7$ означает, что пациент x имеет шанс 70% наличия «злокачественной» ($y = 1$) болезни.

Условия, которые мы вводили ранее:

$$y=1: h_{\theta}(x) \geq 0.5$$

$$y=0: h_{\theta}(x) < 0.5$$

успешно работают при таком виде гипотезы и наглядно интерпретируемы.

Стоит отметить, что не смотря на слово «регрессия» в названии метода, логистическая регрессия не является алгоритмом решения задачи регрессии, а служит для решения задач классификации.

3.3. Граница решения

Значение $h_{\theta}(x) = 0.5$ является пограничным для логистической регрессии. Обычно его относят к классу $y=1$. Интересны значения x , при которых $h_{\theta}(x) = 0.5$. При этом $\theta^T x = 0$. Эти значения отделяют точки, относящиеся к классу ($y=0$) от точек ($y=1$).

Рассмотрим, например, задачу классификации на два класса, в которой каждый объект x описывается двумя признаками: x_1 и x_2 . Для решения этой задачи можно использовать гипотезу вида (g – логистическая функция):

$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2).$$

Предположим, что в результате обучения модели методом градиентного спуска (как это делается рассмотрим позднее) был получен вектор параметров:

$$\theta = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}.$$

Можно построить график функции $\theta^T x = 0$, то есть:

$$-3 + x_1 + x_2 = 0.$$

На рис. 3.5 этот график отображен сиреневой линией, которая отделяет точки класса $y=0$ (синие кружки) от точек класса $y=1$ (красные крестики).

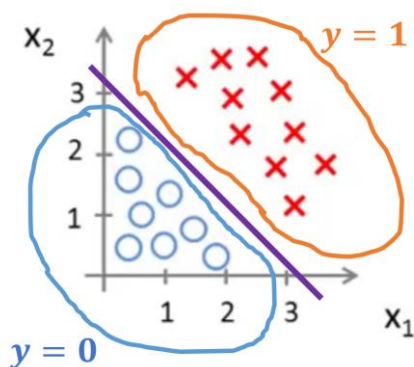


Рис. 3.5

В двумерном случае граница решения представляется достаточно наглядно. В трехмерном случае это может быть достаточно сложная поверхность. В случае большей размерности (когда x описывается десятками и сотнями параметров) представить наглядно границу решения невозможно, но она все равно существует.

3.4. Функция стоимости

Сформулируем постановку задачи логистической регрессии. Пусть дана обучающая выборка из m элементов:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}.$$

Каждый объект x описывается n параметрами x_1, \dots, x_n . Введем фиктивный элемент $x_0 = 1$. Тогда объект x можно представить в виде вектора:

$$x \in \mathbb{R}^{n+1} = \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix}.$$

Ответы $y \in \{0, 1\}$ (два класса).

Гипотеза логистической регрессии:

$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}.$$

Необходимо найти такие параметры θ , при которых достигается «наилучшая классификация». Для определения, какое решение является наилучшим, аналогично линейной регрессии, нам необходимо ввести функцию стоимости.

В случае линейной регрессии функция стоимости имела вид:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2.$$

Однако использовать эту функцию мы не сможем. В линейной регрессии гипотеза имела вид $h_{\theta}(x) = \theta^T x$, и график функции стоимости был гладким и выпуклым (рис. 3.6, а), с четким глобальным минимумом. Алгоритм градиентного спуска легко найдет оптимальные значения θ для такой задачи. Для гипотезы логистической регрессии вида $h_{\theta}(x) = g(\theta^T x)$ график функции стоимости будет иметь вид, подобный рис. 3.6, б с множеством локальных минимумов и не является выпуклой функцией.

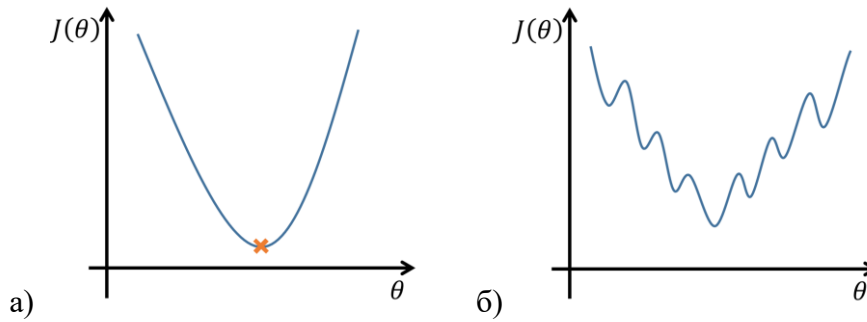


Рис. 3.6

Чтобы избежать этой проблемы, для логистической регрессии используют функцию стоимости другого вида:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^m C(h_{\theta}(x^{(i)}), y^{(i)}),$$

где

$$C(h_{\theta}(x^{(i)}), y^{(i)}) = \begin{cases} -\log(h_{\theta}(x^{(i)})), & \text{если } y = 1; \\ -\log(1 - h_{\theta}(x^{(i)})), & \text{если } y = 0. \end{cases}$$

Рассмотрим это выражение подробнее. Если $y^{(i)} = 1$, то стоимость ошибки для этого объекта $x^{(i)}$ вычисляется по выражению:

$$C(h_{\theta}(x^{(i)}), y^{(i)}) = -\log(h_{\theta}(x^{(i)})).$$

График этой функции приведен на рис. 3.7, а. Если $y^{(i)} = 1$, а $h_{\theta}(x^{(i)}) = 1$, то есть результат классификации правильный, то значение $C(h_{\theta}(x^{(i)}), y^{(i)}) = 0$ (штрафа нет). Если $y^{(i)} = 1$, а $h_{\theta}(x^{(i)}) = 0$, то есть наличие ошибки классификации (или, как еще говорят, ошибки предсказания), $C(h_{\theta}(x^{(i)}), y^{(i)}) \rightarrow \infty$ (большой штраф за ошибку).

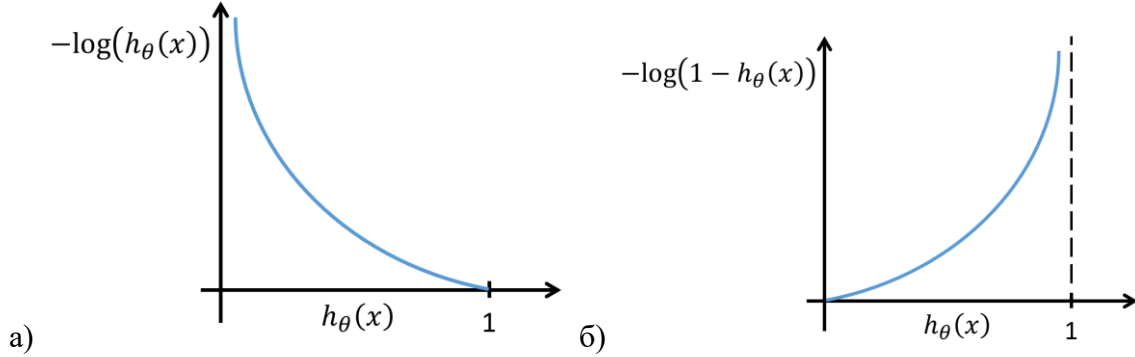


Рис. 3.7

Аналогично происходит и для случая $y^{(i)} = 0$, тогда

$$C(h_{\theta}(x^{(i)}), y^{(i)}) = -\log(1 - h_{\theta}(x^{(i)})),$$

график этой функции приведен на рис. 3.7, б. Как видно, оба варианта являются гладкими и выпуклыми функциями.

Более компактно выражение для $C(h_{\theta}(x^{(i)}), y^{(i)})$ записывается в виде:

$$C(h_{\theta}(x^{(i)}), y^{(i)}) = -y^{(i)} \log(h_{\theta}(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})).$$

Поскольку $y^{(i)} \in \{0, 1\}$ для класса $y^{(i)} = 0$, получаем:

$$C(h_{\theta}(x^{(i)}), y^{(i)}) = -\log(1 - h_{\theta}(x^{(i)})),$$

а для класса $y^{(i)} = 1$:

$$C(h_{\theta}(x^{(i)}), y^{(i)}) = -\log(h_{\theta}(x^{(i)})).$$

Таким образом, окончательно выражение для функции стоимости логистической регрессии принимает вид:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right].$$

Решением задачи логистической регрессии является такой вектор θ^* , при котором $J(\theta)$ принимает минимальное значение. Для нахождения этого решения можно также воспользоваться уже рассмотренным алгоритмом градиентного спуска.

3.5. Обучение логистической регрессии

Оптимизация функции стоимости логистической регрессии (нахождение минимума функции) выполняется аналогично оптимизации множественной линейной регрессии (алгоритм из раздела 2.1). Отличие только в формуле для вычисления $h_{\theta}(x)$.

Для метода применения градиентного спуска необходимо найти выражения для вычисления значения вектора градиента (частные производные по параметрам объекта x) для функции стоимости.

Функция стоимости логистической регрессии:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right].$$

Частные производные для нее будут иметь вид:

$$\frac{\partial}{\partial \theta_0} J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)};$$

$$\frac{\partial}{\partial \theta_1} J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)};$$

...

$$\frac{\partial}{\partial \theta_n} J(\theta) = \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_n^{(i)}.$$

Если мы сравним эти выражения с выражениями для линейной регрессии и алгоритмом из раздела 2.1, то увидим, что они идентичны.

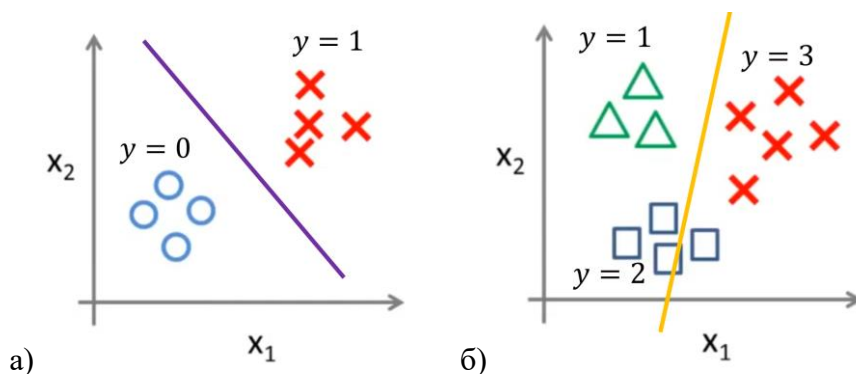
Стоит только иметь в виду, что значение гипотезы, в отличие от линейной регрессии, вычисляется по выражению $h_{\theta}(x) = g(\theta^T x)$.

Таким образом, используя алгоритм из раздела 2.1 и выражение для логистической регрессии, мы можем решать любые задачи классификации на два класса.

3.6. Множественная классификация

Существует множество задач классификации, в которых более двух классов. Например, задача сортировки электронных писем в почтовом ящике, которым необходимо присвоить ярлыки (работа, семья, развлечения, спам и т.д.), предсказание погоды на завтрашний день (ясно, переменная облачность, пасмурно и т.д.).

В разделе 3.3 мы рассматривали границу решения. Для двух классов она достаточно просто определяется (рис. 3.8, а). Если классов больше двух, то найти границу решения будет сложнее (рис. 3.8, б). Гипотеза для такого алгоритма будет уже не прямой линией, найти для нее выражения для частных производных может быть не просто.



а)
Рис. 3.8

б)

Однако, в науке машинного обучения существует подход под названием «один-против-всех», согласно которому задачу классификации на M классов можно решить обучением M различных двухклассовых алгоритмов.

Например, имеется задача классификации на три класса ($y \in \{1,2,3\}$). Выберем один класс (пусть будет $y = 2$) и обозначим его за класс $y = 1$, а все остальные объекты независимо от их принадлежности обозначим за $y = 0$. Обучим один двухклассовый алгоритм для определения вероятности принадлежности объекта x к классу $y = 2$. Затем повторяем этот процесс для классов $y = 1$ и $y = 3$.

Таким образом мы получим M классификаторов, каждый из которых возвращает нам вероятность принадлежности x к одному из классов. Рис. 3.9 наглядно иллюстрирует

ЭТОТ ПОДХОД.

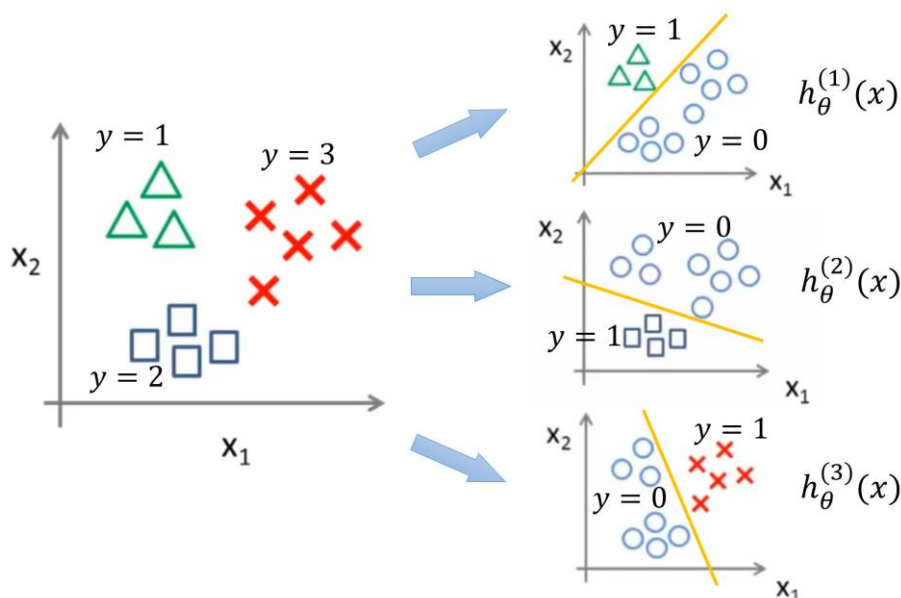


Рис. 3.9

Возьмем произвольный объект x . Чтобы определить, к какому классу он относится, необходимо вычислить значение всех M классификаторов для него. Объект необходимо отнести к тому классу, значение которого максимально (т.е. вероятность принадлежности к этому классу выше всех):

$$H_{\theta}(x) = \underset{k}{\operatorname{argmax}} h_{\theta}^{(k)}(x).$$

Функция argmax возвращает k того $h_{\theta}^{(k)}(x)$, значение которого максимально.

3.7. Продвинутые алгоритмы оптимизации

Оптимизация – математическая задача нахождения минимума функции. В машинном обучении мы используем оптимизацию для нахождения минимума функции стоимости, то есть чтобы определить такие параметры θ , при которых $J(\theta)$ принимает наименьшее значение (рис. 3.10). При этих параметрах мы получаем самое правильное предсказание значений для задачи регрессии и классов для классификации.

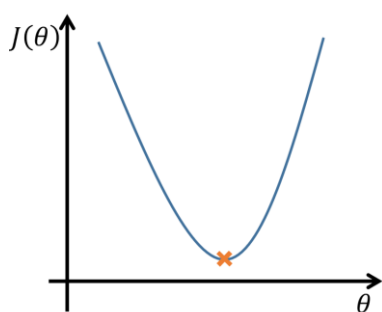


Рис. 3.10

Мы рассмотрели один из самых простых алгоритмов оптимизации – метод градиентного спуска. Его легко запрограммировать, легко понять принципы, на которых он работает. Метод градиентного спуска относится к классу *методов первого порядка*. Эти методы так называются потому, что используют для своей работы только первые частные производные функции (градиент). Методы, которые используют вторые частные производные (гессиан), называют *методами второго порядка*.

Метод градиентного спуска при своей простоте имеет множество недостатков: находит ближайший локальный минимум функции, а не глобальный, выполняет большое число шагов, использует всю обучающую выборку для своей работы. В математике известны десятки более сложных методов оптимизации, которые не обладают такими недостатками.

Если для рассмотренных ранее простейших алгоритмов машинного обучения (линейной и логистической регрессии) использование метода градиентного спуска не критично, то в более сложных алгоритмах, а в особенности для обучения нейронных сетей, используют другие методы оптимизации.

Среди методов первого порядка часто применяются:

- Метод сопряженных градиентов Флетчера – Ривса
- Стохастический градиентный спуск (SGD)
- Адаптивный стохастический градиентный спуск (Adagrad, Adam, RMSProp)

Среди методов второго порядка:

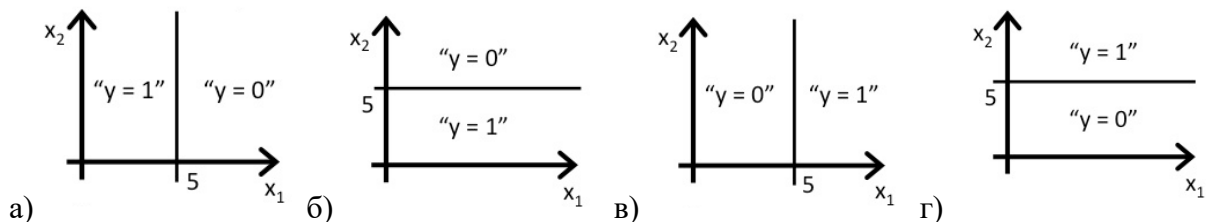
- Алгоритм Бройдена – Флетчера – Гольдфарба – Шанно (BFGS, L-BFGS)
- Метод Ньютона (его тензорные варианты – Shampoo)
- Алгоритм адаптивной оценки Гессияна (AdaHessian)

Эти методы довольно сложны, поэтому их работу мы не будем разбирать. Общий принцип их работы одинаков с градиентным спуском: путем последовательных шагов найти минимум оптимизируемой функции.

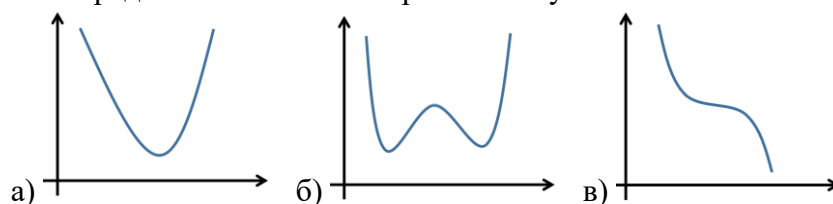
Вопросы и задания

1) Предположим, что имеется задача медицинской диагностики определения доброкачественной или злокачественной раковой опухоли. Для некоторого пациента, описанного вектором параметров x , логистическая регрессия выдала значение $h_\theta(x) = P(y = 1|x; \theta) = 0.7$, в связи с чем мы оцениваем шанс 70%, что опухоль является злокачественной. Какова должна быть оценка, что опухоль является доброкачественной, то есть $P(y = 0|x; \theta)$?

2) Рассмотрим логистическую регрессию с двумя параметрами x_1 и x_2 . Предположим, что $\theta_0 = 5$, $\theta_1 = -1$, $\theta_2 = 0$. Таким образом $h_\theta(x) = g(5 - x_1)$. Что является границей решения для $h_\theta(x)$?



3) Какая из представленных ниже кривых выпуклая?



4) Для логистической регрессии функция штрафа вычисляется по следующему выражению:

$$C(h_{\theta}(x), y) = \begin{cases} -\log(h_{\theta}(x)), & \text{если } y = 1 \\ -\log(1 - h_{\theta}(x)), & \text{если } y = 0 \end{cases}$$

Какие из следующих утверждений верны?

- а) Если $h_{\theta}(x) = y$, то $C(h_{\theta}(x), y) = 0$ (для $y = 0$ и $y = 1$)
- б) Если $y = 0$ и $h_{\theta}(x) \rightarrow 1$, то $C(h_{\theta}(x), y) \rightarrow \infty$
- в) Если $y = 0$ и $h_{\theta}(x) \rightarrow 0$, то $C(h_{\theta}(x), y) \rightarrow 0$
- г) Если $h_{\theta}(x) = 0.5$, то $C(h_{\theta}(x), y) > 0$

5) В задаче классификации на k классов (т.е. $y \in \{1, \dots, k\}$) сколько различных классификаторов методом логистической регрессии необходимо обучить?

4. РЕГУЛЯРИЗАЦИЯ

4.1. Проблема переобучения моделей

Рассмотрим уже известную нам задачу предсказания стоимости дома по его площади, которую мы решаем при помощи линейной регрессии.

Задачу можно попытаться решить самой простой гипотезой вида $h_{\theta}^{(1)}(x) = \theta_0 + \theta_1 x$. В этом случае мы можем получить не очень точное решение. Стоимость дома не линейно зависит от его размера (рис. 4.1, а). Такое решение можно назвать «недообученным». Если обучить модель на обучающей выборке (красные точки на рис.) и предсказать значение для контрольной зеленой точки, то ошибка может быть существенной.

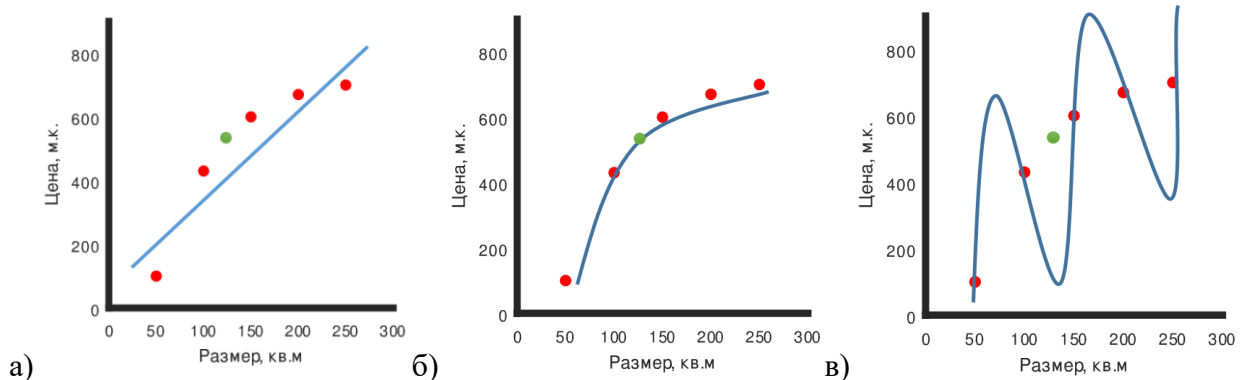


Рис. 4.1

Хорошим решением может оказаться гипотеза $h_{\theta}^{(2)}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$. Обратите внимание, что для ее получения мы воспользовались подходом конструирования переменных (см. разд. 2.3). Ошибка для контрольной точки (зеленая) достаточно мала (рис. 4.1, б). Это пример хорошего решения.

Если в погоне за большей точностью сконструировать еще более сложную гипотезу (рис. 4.1, в): $h_{\theta}^{(3)}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_4 x^4$, можно получить модель, которая очень близко проходит к точкам обучающей выборки, но имеет большую ошибку на данных, которые в обучение не входят (зеленая точка). Эта ситуация имеет название «переобучение», и такая модель не является желательной.

Говорят, что модель имеет хорошую *обобщающую способность*, если хорошо работает и на обучающем наборе, и на новых данных. Переобученные модели, как правило, имеют худшую обобщающую способность, чем недообученные, поэтому с переобучением необходимо бороться.

4.2. Методы борьбы с переобучением

Причинами переобучения моделей являются:

1) Ошибки конструирования модели – слишком сложная функция гипотезы (рис. 4.1, в).

2) Много исходных параметров, когда параметров объекта больше, чем самих объектов в обучающей выборке ($n \gg m$). Можно сформулировать иначе: слишком малый объем обучающей выборки.

Для борьбы с переобучением используются следующие подходы:

1) Использовать внешние критерии оценки $J(\theta)$

2) Уменьшить размер вектора θ

3) Использовать регуляризацию

Первый способ достаточно эффективен и широко распространен, мы рассмотрим его позднее.

Второй подход подразумевает, что необходимо отобрать наиболее важные признаки, или наоборот, удалить лишние. Это можно сделать вручную либо использовать специальные методы отбора признаков, которые мы также будем изучать позднее. В любом случае, подход не всегда применим, поскольку возможны задачи, когда признаки важны все, либо среди них сложно выделить важные/не важные.

Третий подход (*регуляризация*) позволяет сохранить все признаки в исходных данных, но уменьшить для некоторых из них степень влияния на результат. Он хорошо работает, когда имеется множество признаков, все из которых имеют некоторое влияние на результат. Этому подходу и посвящена данная глава.

4.3. Регуляризация

Снова вернемся к тому же примеру из раздела 4.1 и будем использовать гипотезу $h_{\theta}^{(3)}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_4 x^4$. Добавим в функцию стоимости параметры θ_3 и θ_4 с некоторым штрафом (например, 1000). Таким образом, мы будем минимизировать следующую функцию стоимости:

$$J(\theta) = \left[\frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + 1000 \cdot \theta_3^2 + 1000 \cdot \theta_4^2 \right] \rightarrow \min.$$

Это приведет к тому, что величины θ_3 и θ_4 по результату оптимизации будут малы: $\theta_3 \approx 0$, $\theta_4 \approx 0$. Обратите внимание, что в саму гипотезу штрафы на параметры не вносятся, только в функцию стоимости.

Результатом данного приема будет то, что гипотеза $h_{\theta}^{(3)}(x)$ из функции четвертой степени искусственно превращается в квадратичную функцию ($\theta_3 \approx 0$, $\theta_4 \approx 0$). Вместо модели, отображенной на рис. 4.2 синей линией, мы получим модель, отображенную желтой линией. И эта модель будет иметь лучшую обобщающую способность.

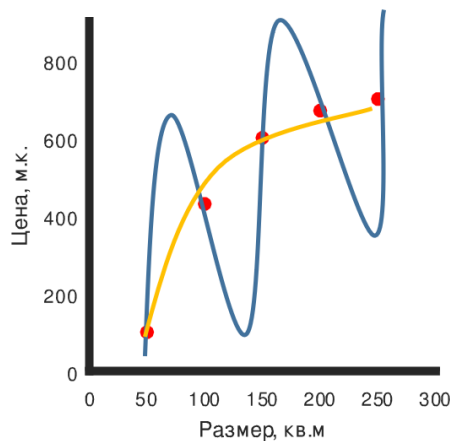


Рис. 4.2

То есть небольшие значения некоторых параметров $\theta_0, \theta_1, \dots, \theta_n$ упрощают гипотезу и делают ее менее склонной переобучению.

В общем виде регуляризованная функция стоимости будет иметь вид:

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right].$$

Параметр λ называется *параметром регуляризации*. Выбор его значения должен быть обоснованным, поскольку, если значение λ очень мало, то эффекта от регуляризации не будет, но если значение λ выбрать очень большим, то модель может получиться очень упрощенной и недообученной.

Обратите внимание, что λ умножается на сумму параметров гипотезы, причем $j =$

$1, \dots, n$, то есть θ_0 не участвует в регуляризации. Можно включать или не включать в регуляризацию θ_0 , результат почти не меняется, он не умножается на признаки объектов. Для упрощения не будем включать θ_0 .

4.4. Регуляризованная линейная регрессия

Чтобы использовать регуляризацию в линейной регрессии, необходимо модифицировать алгоритм градиентного спуска и учесть параметр регуляризации при вычислении частных производных на шаге алгоритма.

Для функции стоимости, заданной

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right],$$

частные производные вычисляются как:

$$\begin{aligned} \frac{\partial}{\partial \theta_0} J(\theta) &= \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)}; \\ \frac{\partial}{\partial \theta_j} J(\theta) &= \left[\frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)} \right] + \frac{\lambda}{m} \theta_j, \quad j = 1, \dots, n. \end{aligned}$$

Заменив выражения в алгоритме из раздела 2.1 на полученные выше, приходим к следующему алгоритму.

Алгоритм градиентного спуска для множественной регрессии с регуляризацией.

Вход: $J(\theta_0, \dots, \theta_n)$ – функция, $\theta_0^0, \dots, \theta_n^0$ – начальные значения переменных, α – темп обучения, ε – критерий остановки, λ – параметр регуляризации.

Выход: $\theta_0^*, \dots, \theta_n^*$ – найденные значения для локального минимума.

Действия:

для $j = 0, \dots, n$:

$$\theta_j := \theta_j^0$$

повторять {

для $j = 0, \dots, n$:

$$d\theta_j := \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\theta_0 := \theta_0 - \alpha d\theta_0$$

для $j = 1, \dots, n$:

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha d\theta_j$$

} пока $d\theta_j > \varepsilon, j = 0, \dots, n$

для $j = 0, \dots, n$:

$$\theta_j^* := \theta_j$$

Конец алгоритма.

В разделе 2.4 мы рассматривали еще один способ решения задачи регрессии – метод нормальных уравнений. Для них тоже существует регуляризации.

Напомним суть метода нормальных уравнений. Если мы имеем обучающую выборку из m элементов, в которой каждый объект описывается n признаками, то можно представить обучающую выборку при помощи матрицы «объекты-признаки» вида:

$$X = \begin{bmatrix} x_0^{(1)} & \dots & x_n^{(1)} \\ \dots & \dots & \dots \\ x_0^{(m)} & \dots & x_n^{(m)} \end{bmatrix} \in \mathbb{R}^{m \times (n+1)}$$

и вектора ответов

$$Y = \begin{bmatrix} y^{(1)} \\ \dots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m.$$

Тогда оптимальное значение вектора параметров θ определяется по выражению:

$$\theta = (X^T X)^{-1} X^T y.$$

Недостатками метода нормальных уравнений является то, что, во-первых, матрица $X^T X$ может быть вырожденной и необратимой, во-вторых, сам процесс нахождения обратной матрицы при больших n может быть очень долгим.

Регуляризованная форма нормального уравнения имеет вид:

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix} \right)^{-1} X^T y,$$

что не многим сложнее нерегуляризованной формы. Обратите внимание, что λ умножается на единичную матрицу, у которой первый элемент равен нулю (поскольку θ_0 мы не регуляризуем). Он может быть равен 1, если θ_0 добавляется в регуляризацию.

Дополнительным бонусом является то, что матрица

$$X^T X + \lambda \begin{bmatrix} 0 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

при $\lambda > 0$ всегда обратима, то есть первый недостаток нормальных уравнений нивелируется. Однако по-прежнему процесс нахождения обратной матрицы будет очень долгим.

4.5. Регуляризованная логистическая регрессия

Регуляризация позволяет повысить обобщающую способность также и логистической регрессии. Рассмотрим задачу классификации раковых заболеваний, исходя из размера опухоли и возраста пациента. На рис. 4.3 зелеными точками показаны доброкачественные опухоли ($y=0$), а красными злокачественные ($y=1$).

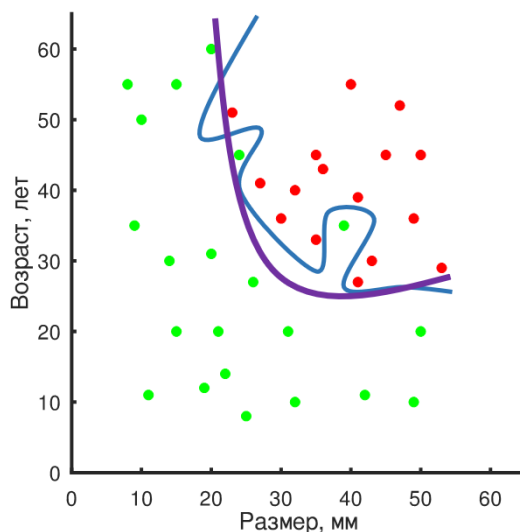


Рис. 4.3

Сложная гипотеза, например, следующего вида

$$h_{\theta}(x) = g(\theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_4 x^4)$$

может приводить к переобучению и формировать модель, как показано синей линией на рис.

Для регуляризации логистической регрессии в функцию стоимости вводится параметр регуляризации:

$$J(\theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2.$$

Аналогично тому, как это было для нерегуляризованной версии, алгоритм градиентного спуска для регуляризованной логистической регрессии аналогичен алгоритму регуляризованной линейной регрессии (т.е. алгоритму из раздела 4.4). Разница только в том, что гипотеза $h_{\theta}(x)$ вычисляется с использованием логистической функции.

Вопросы и задания

1) Рассмотрим задачу классификации. Если гипотеза $h_{\theta}(x)$ переобучается на наборе данных, то это означает, что ...

а) гипотеза делает точные предсказания на обучающем наборе данных и хорошо обобщается, делает точные предсказания для новых, ранее не встречаемых примерах.

б) гипотеза делает плохие предсказания на обучающем наборе данных, но хорошо обобщается.

в) гипотеза делает точные предсказания на обучающем наборе данных, но плохо обобщается.

г) гипотеза делает плохие предсказания на обучающем наборе данных и плохо обобщается.

2) Решая задачу регуляризованной линейной регрессии градиентным спуском, мы должны получить значения θ такие, чтобы найти минимум функции стоимости. Что если задать значение λ очень большим (например, $\lambda = 10^{10}$) ?

а) Алгоритм отрабатывает правильно, большое значение λ не повредит ему

б) Устранить переобучение не получится, но на обучающем наборе данных результат будет достаточно точным

в) Получим недообученную модель

г) Градиентный спуск не сможет завершить работу

3) Решаем задачу регуляризованной линейной регрессии градиентным спуском на обучающем наборе размером $m > 0$, используя некоторый небольшой шаг $\alpha > 0$ и некоторый заданный параметр регуляризации $\lambda > 0$. Для вычисления следующего значения $\theta_1, \dots, \theta_n$ используется выражение:

$$\theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m} \right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

Какое из следующих утверждений является верным?

а) $1 - \alpha \frac{\lambda}{m} > 1$

б) $1 - \alpha \frac{\lambda}{m} = 1$

в) $1 - \alpha \frac{\lambda}{m} < 1$

5. НЕЙРОННЫЕ СЕТИ

5.1. Проблемы сложной нелинейной классификации

В разделе 3 мы познакомились с методом классификации, который называется логистической регрессией. Представим, что нам необходимо решить задачу классификации со сложной нелинейной гипотезой, например такой, чтобы граница решения принимала вид, как показано на рис. 5.1.

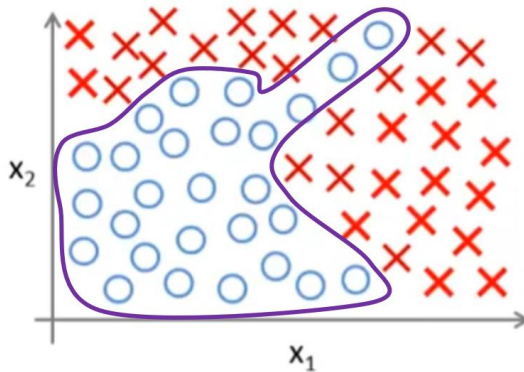


Рис. 5.1

Подобную границу решения может иметь гипотеза вида

$$h_{\theta}(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1 x_2 + \dots + \theta_8 x_2^2 + \theta_{15} x_1 x_2^4 + \dots)$$

Напомним, что g – логистическая функция (сигмоида) вида:

$$g(z) = \frac{1}{1 + e^{-z}}$$

Обучить такую гипотезу не представляет сложности, но в реальности задачи редко ограничиваются только двумя параметрами x_1, x_2 . Задача может иметь сотни, тысячи и более признаков у объекта x , что для подобного вида гипотез выливается в огромное число параметров. Для гипотезы в виде квадратичной функции количество параметров модели пропорционально n^2 , где n – количество признаков. Для кубической гипотезы количество параметров модели пропорционально n^3 и т.д. Например, если $n = 100$, кубическая гипотеза будет иметь примерно 170 тысяч параметров θ .

Для задачи классификации полутоновых изображений размером 50×50 пикселей кубическая гипотеза будет иметь порядка 15 млн параметров. Для изображений 500×500 их количество возрастет до $15 \cdot 10^{15}$.

Очевидно, что обучение такой гипотезы – слишком трудоемкая задача. Для ее упрощения прибегают к различным методам. Например, для изображений используют различные преобразования: метод главных компонент, преобразование Фурье, каскады фильтров Хаара и т.д.

Еще одним хорошим и эффективным способом решения сложной нелинейной задачи с большим числом признаков являются искусственные нейронные сети.

5.2. Искусственные нейронные сети

Теория искусственных нейронных сетей появилась в 1940-е годы, но не получила практического применения из-за слабого развития вычислительной техники и к 60-м интерес к ней угас. Всплеск внимания к нейронным сетям возник в 1980-е, когда персональные компьютеры широко распространились в быту и научной сфере. Но настоящее развитие нейронные сети получили, начиная примерно с конца 90-х годов 20 века, когда был предложен эффективный алгоритм обучения, и уже в нашем тысячелетии.

В настоящее время с использованием нейронных сетей решается широкий спектр задач, таких как распознавание, классификация, предсказание, обработка естественного

языка, генерация изображений, музыки и многое другое.

На рисунке 5.2 показан модель нейрона, который называется *персептроном*.

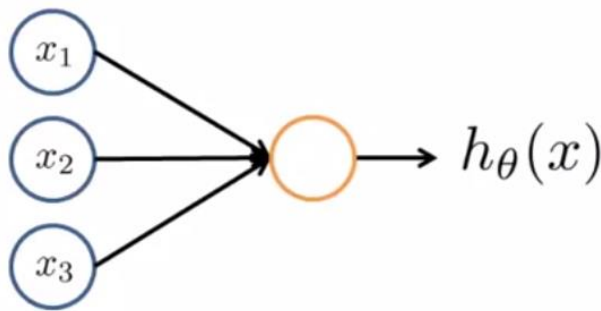


Рис. 5.2

С левой части отображены входы $x = [x_1, x_2, \dots, x_n]$, n – размер вектора признаков. Над входными значениями выполняется вычисление линейного полинома:

$$z = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

Вектор $\theta = [\theta_0, \theta_1, \theta_2, \dots]$ называется параметрами или *весами* нейрона.

К результату вычисления полинома применяется *функция активации* нейрона. Известно множество функций активации: линейная, гиперболическая и т.д. Мы используем сейчас сигмоидную (логистическую) функцию активации, известную по логистической регрессии:

$$g(z) = \frac{1}{1 + e^{-z}}$$

Таким образом, результат вычисления нейрона определяется как:

$$h_\theta(x) = g(\theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots)$$

Здесь мы добавили к входным параметрам $x_0 = 1$ аналогично тому, как это выполнялось в разделе 3.

Нейроны можно соединять друг с другом, объединяя в нейронную сеть. В этом случае результаты вычисления одних нейронов подаются в качестве входов на другие нейроны.

На рис. 5.3 показан пример нейронной сети.

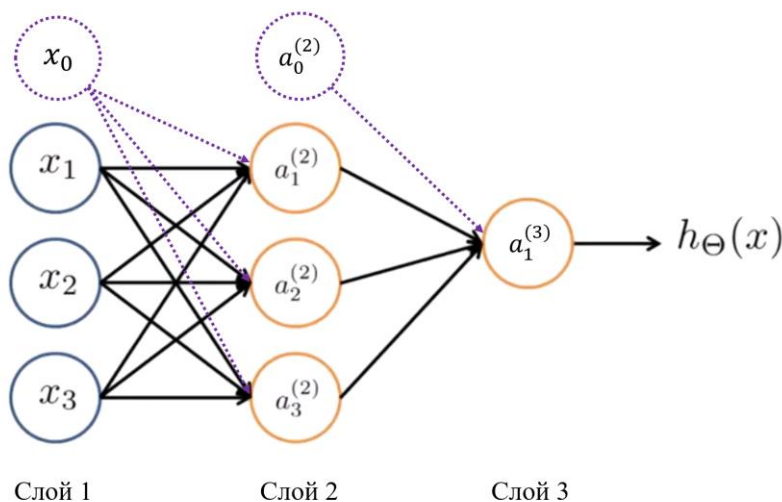


Рис. 5.3

В нейронной сети выделяют слои:

Входной слой – представляет собой значения входного вектора признаков;

Выходной слой – значение одного (возможно, нескольких) выходной нейронов;

Скрытые слои – все остальные слои между первым и последним.

Количество нейронов в j -м слое будем обозначать S_j . Значения, вычисляемые i -м нейроном j -го слоя будем обозначать $a_i^{(j)}$. Это значения также называют *активациями* нейронов (иногда *откликами*).

Для входного слоя нейросети вычислений не производится:

$$a_1^{(1)} = x_1; a_2^{(1)} = x_2; \dots; a_n^{(1)} = x_n.$$

Также в сети неявно присутствует фиктивный элемент $a_0^{(1)} = x_0 = 1$, но зачастую его не отображают на рисунке.

Введем понятие *матрицы весов* между j -м и $(j+1)$ -м слоем:

$$\Theta^{(j)} = \begin{bmatrix} \Theta_{10}^{(j)} & \Theta_{11}^{(j)} & \dots & \Theta_{1S_j}^{(j)} \\ \Theta_{20}^{(j)} & \Theta_{21}^{(j)} & \dots & \Theta_{2S_j}^{(j)} \\ \dots & \dots & \dots & \dots \\ \Theta_{S_{j+1}0}^{(j)} & \Theta_{S_{j+1}1}^{(j)} & \dots & \Theta_{S_{j+1}S_j}^{(j)} \end{bmatrix}$$

Размер этой матрицы составляет $S_{j+1} \times (S_j + 1)$. Например, для сети на рисунке 5.3 размер матрицы весов между первым и вторым слоем 3×4 , а между слоями 2 и 3 матрица имеет размер 1×4 .

Вычисление активации одного нейрона в сети выполняется по формуле:

$$a_i^{(j+1)} = g \left(\theta_{i0}^{(j)} a_0^{(j)} + \theta_{i1}^{(j)} a_1^{(j)} + \dots + \theta_{iS_j}^{(j)} a_{S_j}^{(j)} \right), \quad a_0^{(j)} = 1.$$

Это выражение можно компактно записать в матричной форме:

$$a^{(j+1)} = g \left(\theta^{(j)} a^{(j)} \right)$$

Здесь $a^{(j)} = (a_0^{(j)}, a_1^{(j)}, \dots, a_{S_j}^{(j)})$, $a_0^{(j)} = 1$.

Как видно, сам по себе нейрон очень прост и сильно напоминает простейшую логистическую линейную регрессию. Преимущества нейронных сетей проявляются в использовании большого количества нейронов, объединения их в сложные сети из множества слоев, взаимосвязь которых определяется *архитектурой нейронной сети*.

Архитектура определяет:

- Количество нейронов входного слоя;
- Количество нейронов выходного слоя;
- Количество слоев сети;
- Типы слоев (плотный слой, аналогичный рассмотренному в данном разделе, сверточный слой и другие);
- Типы связей в слоях (прямые или обратные, характерные для рекуррентных нейронных сетей);
- Типы функций активации (различают сигмоидную, линейную, гиперболическую и другие).

В данном разделе мы ознакомились только с одним типом нейронных сетей из плотных слоев персептронов (когда все нейроны в слое равнозначны и каждый нейрон предыдущего слоя связан с каждым нейроном следующего слоя). Однако, в мире существует большое количество других более сложных архитектур нейронных сетей.

5.3. Алгоритм прямого распространения

Как показано в предыдущем пункте, активацию всего слоя нейронной сети можно вычислить достаточно быстро с использованием матричного перемножения. Это приводит нас к следующему достаточно простому и быстрому алгоритму, который получил название *алгоритма прямого распространения*.

Алгоритм прямого распространения.

Вход: $x = [x_1, \dots, x_n]$ – вектор признаков;

$\Theta = [\Theta^{(1)}, \Theta^{(2)}, \dots, \Theta^{(N-1)}]$ – матрицы весов слоев нейронной сети;

N – количество слоев сети.

Выход: $h_{\theta}(x)$ – активация нейронной сети.

Начало:

$$a^{(1)} = [1 \quad x_1 \quad x_2 \quad \dots \quad x_{s_1}]^T$$

для $j = 1 \dots (N - 1)$ выполнить:

$$z^{(j+1)} = \theta^{(j)} a^{(j)}$$

$$a^{(j+1)} = [1 \quad | \quad g(z^{(j+1)})]^T$$

конец

$$h_{\theta}(x) = a^{(N)}.$$

Конец алгоритма.

В алгоритме используется операция конкатенации векторов. Конкатенацией называют операцию слияния матриц вдоль одной из сторон при условии, что размеры по этой стороне у них совпадают. Например, даны два вектора: $a = [a_1 \quad a_2 \quad a_3]$ и $b = [b_1 \quad b_2]$. Их конкатенацией называется $[a \mid b] = [a_1 \quad a_2 \quad a_3 \quad b_1 \quad b_2]$. В данном алгоритме мы использовали эту операцию, чтобы добавить фиктивные единицы к векторам.

Выход алгоритма $h_{\theta}(x)$ может быть одним числом, если в выходном слое всего один нейрон, но также и вектором (или в некоторых конфигурациях нейросетей матрицей), если нейронов больше.

Алгоритм записан так, что на его вход подается один объект x . Но можно легко использовать этот алгоритм и для вычисления активации нейросети в ответ сразу на несколько объектов. В этом случае на вход сети подается не вектор x , а матрица:

$$X = \begin{bmatrix} x_1^{(1)} & \dots & x_n^{(1)} \\ \dots & \dots & \dots \\ x_1^{(m)} & \dots & x_n^{(m)} \end{bmatrix},$$

где m – количество объектов (размер выборки), n – как и раньше, размер вектора признаков объекта. Матрицу X называют *матрицей признаков объектов*.

Достоинством матричной реализации алгоритма прямого распространения является возможность его быстрого вычисления с помощью многоядерных матричных процессоров (как, например, графические процессоры или специальные вычислители).

5.4. Реализация логических функций

В данном разделе рассмотрим несколько примеров нейронных сетей для вычисления булевых функций. Булевы (логические) функции достаточно легко реализуются на нейросетях и позволяют наглядно продемонстрировать процесс вычисления.

Рассмотрим булеву операцию И (AND, \wedge , &). Таблица истинности операции имеет вид:

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

Значения $x_1, x_2 \in \{0, 1\}$, и значение $(x_1 \wedge x_2) \in \{0, 1\}$ тоже. Однако, нейрон с сигмоидной функцией активации возвращает значения в интервале $(0, 1)$. То есть мы можем говорить только о приближенном вычислении логической функции с помощью нейронной сети. На рис. 5.4 показана архитектура такой сети всего из одного нейрона.

Если записать активацию этого нейрона в виде выражения, получим:

$$h_{\theta}(x) = g(-30 + 20x_1 + 20x_2)$$

На рисунке 5.5 показан график сигмоидной функции активации:

$$g(z) = \frac{1}{1 + e^{-z}}$$

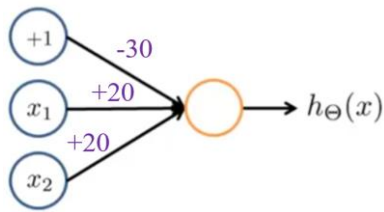


Рис. 5.4

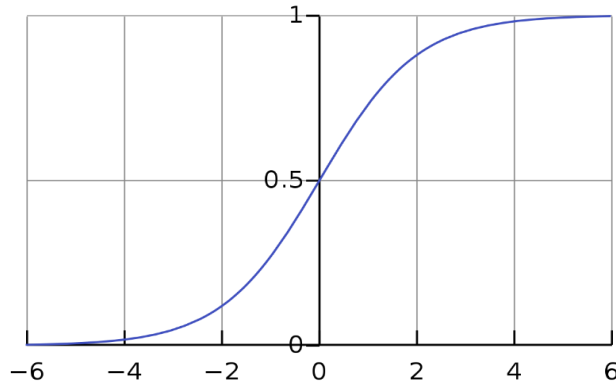


Рис. 5.5

Хорошо видно, что при значениях аргумента, больших 6, результат функции практически равен 1, а при значениях аргумента меньше -6, он практически равен 0.

В таком случае, если подставить в выражение значения $x_1 = 0, x_2 = 0$, то $h_\theta(x) = g(-30) \approx 0$. При $x_1 = 1$ или $x_2 = 1$ значение $h_\theta(x) = g(-10) \approx 0$. А при $x_1 = 1$ одновременно с $x_2 = 1$ значение $h_\theta(x) = g(10) \approx 1$.

Если представить значения $h_\theta(x)$ от разных аргументов в виде таблицы, получим:

x_1	x_2	$h_\theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(+10) \approx 1$

Как видим, значения $h_\theta(x)$ повторяют таблицу истинности булевой функции И, так что можно считать, что $h_\theta(x) \approx x_1 \wedge x_2$.

Таким же образом можно реализовать другие булевы функции, например, логическое ИЛИ (OR, $\vee, |$). На рис. 5.6 показана нейросеть, которая ее вычисляет.

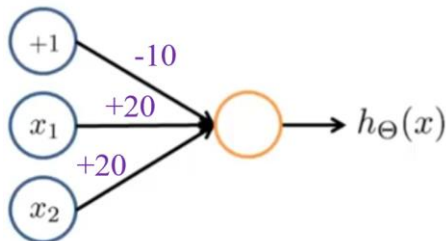


Рис. 5.6

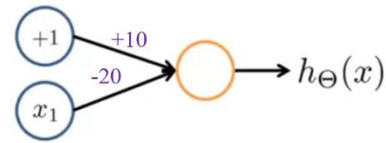
Результат вычисления активации нейросети в сравнении с таблицей истинности:

x_1	x_2	$h_\theta(x)$	$x_1 \vee x_2$
0	0	$g(-10) \approx 0$	0
0	1	$g(+10) \approx 1$	1
1	0	$g(+10) \approx 1$	1
1	1	$g(+30) \approx 1$	1

На рис. 5.7 показана нейросеть для вычисления логического отрицания НЕ (NOT, \neg) и таблица вычисления активации нейросети в сравнении с таблицей истинности функции НЕ.

x_1	$h_\theta(x)$	x_1
0	$g(+10) \approx 1$	1
1	$g(-10) \approx 0$	0

Рис. 5.7



Эти три нейрона (рис. 5.4, 5.6, 5.7) можно комбинировать для создания более сложной булевой функций или даже выражений.

Покажем, как с помощью их комбинации можно построить нейросеть для вычисления булевой операции ИСКЛЮЧАЮЩЕЕ ИЛИ-НЕ (XNOR). Значение этой функции равно 1, когда аргументы равны, и 0, когда они различаются. Структура нейросети и таблица истинности показана на рис. 5.8. Она скомбинирована из операции И (нейрон $a_1^{(2)}$), операции ИЛИ-НЕ (нейрон $a_2^{(2)}$), операции ИЛИ (нейрон $a_1^{(3)}$).

Операция ИЛИ-НЕ (отрицание ИЛИ) может быть вычислена как $\neg(x_1 \vee x_2) = \neg x_1 \wedge \neg x_2$.

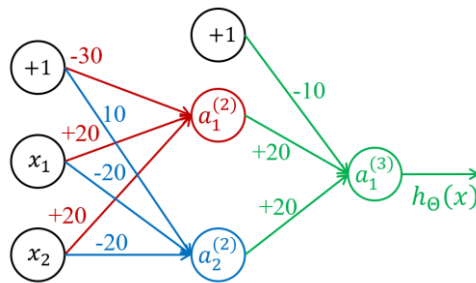


Рис. 5.8

x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

5.5. Множественная классификация

Выход нейронной сети $h_\theta(x)$ с сигмоидной функцией активации имеет значение в интервале от 0 до 1, также как и для логистической регрессии. Аналогично, его можно рассматривать как вероятность принадлежности поданного на вход объекта x какому-либо классу. Например, если на вход нейросети подаются изображения автомобилей, а саму нейросеть обучили определять изображения автомобилей, то $h_\theta(x)$ можно интерпретировать как вероятность, что на поданном изображении есть автомобиль: 1 – точно есть, 0 – точно нет, 0.6 – скорее есть, с вероятностью 60%.

Если перед нами стоит задача классификации не на два класса, а более, то в разделе 3.6 мы разобрали подход «один-против-всех», который позволяет ее решить. Данный подход годится и для нейронных сетей, несколько отличается только его реализация. Если для логистической регрессии для классификации на M классов нам было необходимо обучить M независимых логистических модели, то для нейронной сети процесс обучения остается один, но в нейросети становится M нейронов выходного слоя, соответственно $h_\theta(x)$ представляет из себя вектор $h_\theta(x) = [h_1, h_2, \dots, h_M]$. Значения h_i этого вектора интерпретируются как вероятности соотнесения объекта x с классом i . Для окончательного решения, к какому конкретному классу k отнести x , вычисляется:

$$k = \operatorname{argmax} h_\theta(x)$$

Напомним, что функция argmax для вектора возвращает индекс (позицию) элемента с максимальным значением. То есть, в данном случае, индекс класса с наибольшей вероятностью.

Например, перед нами задача классификации изображения на четыре класса: автомобили ($k = 1$), пешеходы ($k = 2$), велосипеды ($k = 3$), паровозы ($k = 4$).

Отклик (активация) нейросети составляет:

$$h_{\theta}(x) = \begin{bmatrix} 0.94 \\ 0.03 \\ 0.09 \\ 0.20 \end{bmatrix}$$

Тогда $k = \arg\max h_{\theta}(x) = 1$, так как значение в первой позиции вектора максимально. Понимать вектор $h_{\theta}(x)$ мы можем таким образом, что изображение с вероятностью 94% содержит автомобиль, 3% на то, что это пешеход, 9% вероятности, что это велосипед, и вероятность 20%, что это паровоз. Очевидно, что окончательное решение остается за автомобилем, так как вероятность там самая высокая.

5.6. Функция стоимости нейронной сети

Прежде чем обучать нейронные сети необходимо определиться с функцией стоимости, которая позволит нам оценивать качество работы нейронной сети.

Рассмотрим нейронную сеть, которая состоит из множества входных нейронов, множества выходных нейронов и множества скрытых слоев, например, как показано на рис. 5.9.

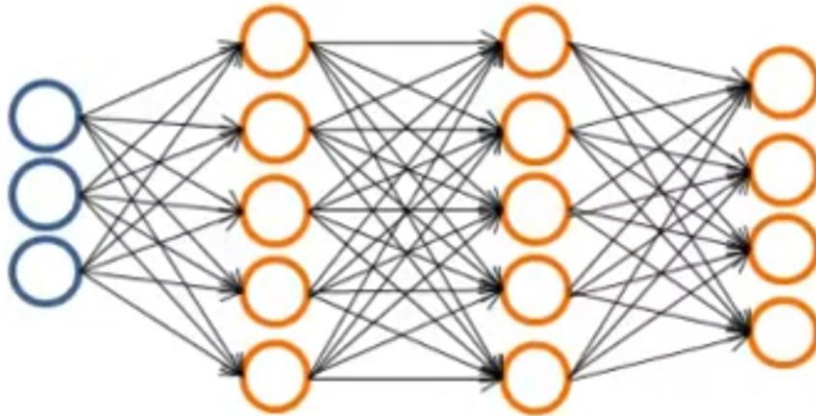


Рис. 5.9

L – количество слоев в сети (на рис. 5.9 $L=4$); s_l – количество нейронов в слое l ($1 \leq l \leq L$), фиктивный +1 не учитывается. На рис. 5.9: $s_1 = 3$, $s_2 = 5$, $s_3 = 5$, $s_4 = 4$.

Пусть задана обучающая выборка из m пар:

$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}.$$

Каждый $x^{(i)}$ является вектором из s_1 элементов:

$$x^{(i)} \in \mathbb{R}^{s_1}.$$

Каждый $y^{(i)}$ является вектором из s_L элементов:

$$y^{(i)} \in \mathbb{R}^{s_L}.$$

Тогда функция стоимости нейронной сети с регуляризацией может быть определена как:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \sum_{k=1}^{s_L} \left[y_k^{(i)} \log \left(\left(h_{\theta}(x^{(i)}) \right)_k \right) + \left(1 - y_k^{(i)} \right) \log \left(1 - \left(h_{\theta}(x^{(i)}) \right)_k \right) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(\theta_{i,j}^{(l)} \right)^2,$$

где $\left(h_{\theta}(x^{(i)}) \right)_k$ – отклик k -го нейрона выходного слоя, для вычисления которого используется алгоритм прямого распространения.

Минимизируя функцию $J(\theta)$ по всем весам нейронной сети θ (это $L - 1$ матриц) мы

обучим нейронную сеть. Функция $J(\theta)$ для нейронной сети не является выпуклой, она достаточно сложная, поэтому для ее оптимизации используется какой-либо из продвинутых алгоритмов, перечисленных в разделе 3.7, например, стохастического градиентного спуска.

Но для использования метода оптимизации необходимо найти способ вычисления частных производных параметров θ (всех, между каждой парой нейронов каждых двух соседних слоев сети), что является очень сложной задачей. Для нахождения частных производных используется алгоритм обратного распространения.

5.7. Алгоритм обратного распространения

Задачей алгоритма является вычисление частных производных

$$\frac{\partial}{\partial \theta_{i,j}^{(l)}} J(\theta).$$

$a_j^{(l)}$ – активация (результат вычисления) j -го нейрона слоя l . Обозначим $\delta_j^{(l)}$ – «ошибка» j -го нейрона слоя l . Для выходного слоя определить ошибку очень просто:

$$\delta^{(L)} = a^{(L)} - y.$$

Для скрытого слоя нейронной сети:

$$\delta^{(i)} = (\theta^{(i)})^T \delta^{(i+1)} \cdot g'(z^{(i)}),$$

где

$$g'(z) = g(z)(1 - g(z))$$

– производная сигмоидной функции активации:

$$g(z) = \frac{1}{1 + e^{-z}}.$$

Для входного слоя нейронной сети ошибка не вычисляется.

Частные производные определяются как:

$$\frac{\partial}{\partial \theta_{i,j}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)}.$$

Или в матричной форме:

$$\frac{\partial}{\partial \theta^{(l)}} J(\theta) = \delta^{(l+1)} (a^{(l)})^T.$$

Поскольку $\delta^{(l+1)}$ – вектор длиной s_{l+1} , а $a^{(l)}$ – вектор длиной $s_l + 1$, то между слоями (l) и $(l + 1)$ мы получим матрицу частных производных размером $(s_{l+1}) \times (s_l + 1)$, таким же, как и матрица параметров $\theta^{(l)}$.

Целиком алгоритм обратного распространения следующий.

Алгоритм обратного распространения.

Вход: Обучающий набор $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$; веса сети: $\theta = (\theta^{(1)}, \dots, \theta^{(L-1)})$; параметр регуляризации λ .

Выход: Значения частных производных: $D^{(1)}, \dots, D^{(L-1)}$.

Действия:

$$\Delta_{ij}^{(l)} = 0 \quad (\text{для } \forall l, i, j)$$

для $i = 0, \dots, m \{$

$$a^{(1)} = x^{(i)}$$

$$a^{(2)}, \dots, a^{(L)}, z^{(2)}, \dots, z^{(L)} = FP(x^{(i)}, \theta)$$

$$\delta^{(L)} = a^{(L)} - y^{(i)}$$

$$\delta^{(l)} = (\theta^{(l)})^T \delta^{(l+1)} \cdot g'(z^{(l)}), \quad l = L - 1, \dots, 2$$

$$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$$

}

$$D_{ij}^{(l)} = \begin{cases} \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \theta_{ij}^{(l)}, & j \neq 0 \\ \frac{1}{m} \Delta_{ij}^{(l)}, & j = 0 \end{cases}$$

Конец алгоритма.

Искомые частные производные хранятся в матрицах D :

$$\frac{\partial}{\partial \theta_{i,j}^{(l)}} J(\theta) = D_{ij}^{(l)}.$$

Эти частные производные можно использовать на шаге любого градиентного метода оптимизации для нахождения минимума функции стоимости.

Стоит только учесть один нюанс. Методы оптимизации принимают вектор частных производных (градиент), в то время как алгоритм обратного распространения возвращает $L - 1$ матриц $D^{(1)}, \dots, D^{(L-1)}$. Поэтому элементы всех матриц поочередно записываются в один длинный вектор, который передается процедуре оптимизации. Процедура не знает, сколько слоев в нейронной сети, сколько в каждом слое весов. Вся нейронная сеть для нее представляется одним длинным вектором параметров. После выполнения процедуры элементы в том же порядке извлекаются из вектора и формируют нужные матрицы.

5.8. Численная оценка градиента

Еще одним способом вычисления частной производной для нейронной сети является численная оценка градиента. Из высшей математики известно, что производная гладкой функции в заданной точке равно тангенсу угла касательной к функции в данной точке (рис. 5.10).

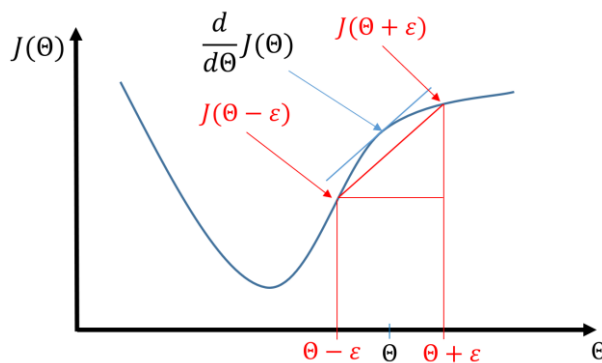


Рис. 5.10

Если значение производной $J'(\theta)$ в точке θ не известно (например, выражение для его вычисления нельзя найти аналитически), но известно значение самой функции $J(\theta)$ в любой точке (как правило, его можно вычислить), то приближенное значение производной можно найти как:

$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon}$$

при достаточно малом ε ($\varepsilon = 10^{-4} \dots 10^{-6}$). Чем меньше ε , тем точнее.

Недостатком численной оценки градиента является то, что для реального обучения нейронной сети его использование слишком медленно по сравнению с алгоритмом обратного распространения.

Обычно численную оценку градиента применяют для проверки правильности обучения нейронной сети:

- 1) Вычисляют $D_{ij}^{(l)}$ (алгоритмом обратного распространения)
- 2) Вычисляют $N_{ij}^{(l)} = \frac{J(\theta + \varepsilon) - J(\theta - \varepsilon)}{2\varepsilon}$

3) Если $D_{ij}^{(l)} \approx N_{ij}^{(l)}$, значит алгоритм отрабатывает верно
После проверки численный расчет отключают и не используют.

5.9. Начальная инициализация весов

При рассмотрении линейной или логистической регрессии мы могли инициализировать начальные значения параметров модели любыми значениями, в том числе нулевыми. При обучении нейронной сети инициализировать веса нулевыми значениями нельзя, также нежелательно использовать любые одинаковые значения.

Это связано с особенностью работы алгоритма обратного распространения.

Если начальные веса нейронов нулевые, это приведет к тому, что отклики нейронов в слое сети будут одинаковыми:

$$a_i^{(l)} = a_j^{(l)}.$$

Следовательно, будут одинаковыми и значения ошибок:

$$\delta_i^{(l)} = \delta_j^{(l)}.$$

В свою очередь, будут одинаковыми значения частных производных:

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = \frac{\partial}{\partial \theta_{km}^{(l)}} J(\theta).$$

В конце это приведет к тому, что будут одинаковыми значения весов:

$$\theta_{ij}^{(l)} = \theta_{km}^{(l)}.$$

Это называют «*проблемой симметрии*» – когда многие (или все) нейроны имеют одинаковый отклик.

Способ избавления от этой проблемы прост: все веса нейронной сети инициализируются случайными значениями.

5.10. Обучение нейронной сети

Подытожим все, что было рассмотрено в данной главе и сформулируем на основе всех рассмотренных выражений и алгоритмов общий процесс обучения нейронной сети.

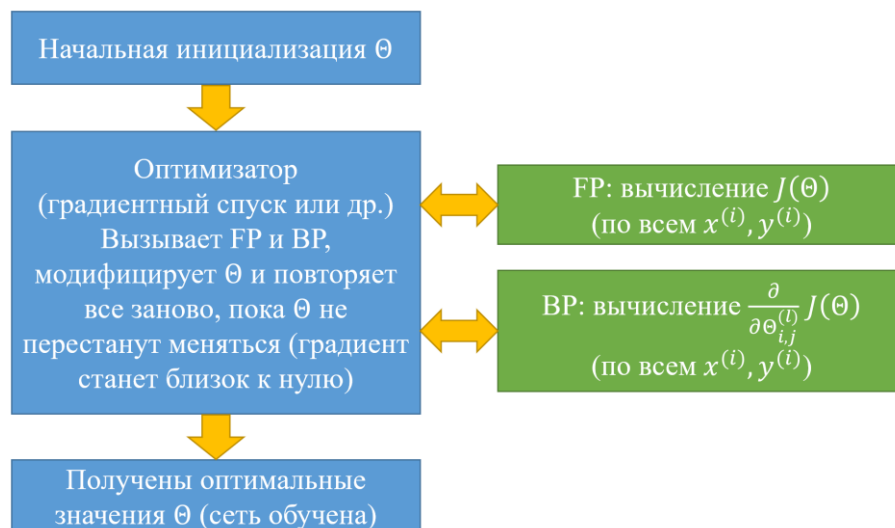


Рис. 5.11

В общем случае, для обучения сети необходимо следующее:

1. Выполнить начальную инициализацию весов θ случайными значениями
2. Реализовать алгоритм прямого распространения (ФР) для вычисления $h_{\theta}(x^{(i)})$ для любого входного $x^{(i)}$
3. Реализовать функцию вычисления $J(\theta)$

4. Реализовать алгоритм обратного распространения (БР) для вычисления частных производных $\frac{\partial}{\partial \theta_{i,j}^{(l)}} J(\theta)$
5. (Опционально) Реализовать численную оценку градиента и проверить правильность работы алгоритма БР
6. Использовать продвинутый алгоритм оптимизации для нахождения оптимальных θ

При этом алгоритм оптимизации на каждом своем шаге вызывает поочередно функции прямого распространения (для вычисления отклика нейронной сети на обучающую выборку) и функцию обратного распространения (для вычисления градиента весов сети), как показано на рис. 5.11.

Вопросы и задания

1) Дана нейронная сеть из трех слоев, входной слой содержит 2 нейрона, скрытый слой содержит 4 нейрона, выходной слой содержит 1 нейрон. Определите, какой размер имеет матрица $\theta^{(1)}$.

2) Основываясь на алгоритме прямого распространения, какой из представленных ниже вариантов вычисления $a^{(2)}$ является верным?

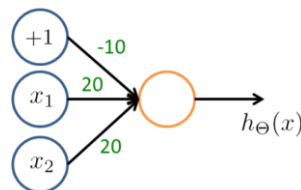
а) $a^{(2)} = \theta^{(1)} a^{(1)}$

б) $z^{(2)} = \theta^{(2)} a^{(1)}; a^{(2)} = [1 \mid g(z^{(2)})]$

в) $z^{(2)} = \theta^{(1)} a^{(1)}; a^{(2)} = [1 \mid g(z^{(2)})]$

г) $z^{(2)} = \theta^{(1)} g(a^{(1)}); a^{(2)} = g(z^{(2)})$

3) Какую булеву функцию приближенно вычисляет нейросеть на данном рисунке? ($x_1, x_2 \in \{0, 1\}$)



а) $x_1 \wedge x_2$

б) $x_1 \vee x_2$

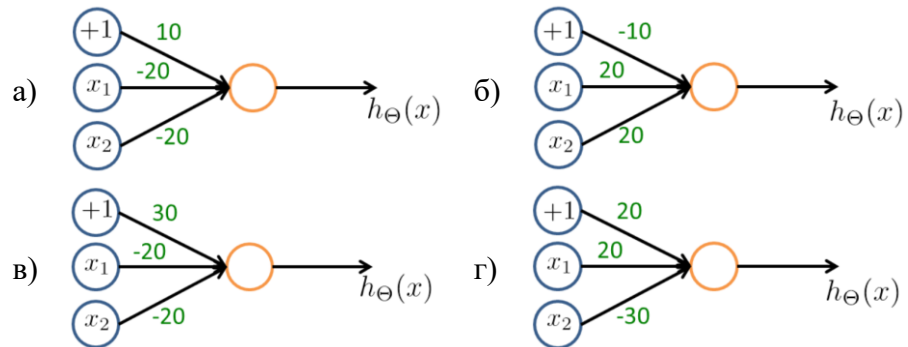
в) $\neg(x_1 \vee x_2)$

г) $\neg(x_1 \wedge x_2)$

4) Пусть $x_1, x_2 \in \{0, 1\}$. Мы хотим приближенно вычислить булеву функцию:

$$y = \neg x_1 \& \neg x_2$$

Какая из представленных нейронных сетей решает эту задачу?



5) В задаче классификации на 10 классов мы используем нейронную сеть с тремя слоями. Скрытый (второй) слой имеет 5 нейронов. Если используется подход «один-против-всех», сколько элементов содержит матрица $\theta^{(2)}$?

б) Рассмотрим следующую процедуру начальной инициализации параметров нейронной сети:

Шаг 1. Вычислим число $r = rand(-\tau, \tau)$

Шаг 2. $\theta_{i,j}^{(l)} = r$ для $\forall l, i, j$

Будет ли такая начальная инициализация удачной?

- а) Да, параметры заданы случайно
- б) Да, если $r \neq 0$
- в) Нет, это создает проблему симметрии
- г) Возможно, зависит от входных x

6. ДИАГНОСТИКА МАШИННОГО ОБУЧЕНИЯ

6.1. Методы улучшения обобщающей способности модели

Предположим, мы построили модель линейной регрессии (как в главе 2), используя обучающую выборку, нашли оптимальный вектор θ . Представим, что после этого к нам попали новые данные, и результаты, которые выдает наша гипотеза с этими данными плохо совпадают. То есть хорошо работающая на обучающем наборе модель плохо работает на новых данных. В главе 4 способность модели хорошо работать на обучающей выборке и на новых данных мы называли обобщающей. Получается, что наша модель имеет плохую обобщающую способность. Какие есть способы улучшения обобщающей способности:

- 1) *Увеличить обучающий набор данных.* Мы можем добавить в обучающую выборку новые данные и заново обучить модель. В зависимости от задачи, источниками новых данных могут быть результаты экспериментов, наблюдений, опросов, экспертные мнения, статистические данные и т.д. В некоторых случаях может потребоваться очень много времени и затрат на получение новых данных.
- 2) *Уменьшить число используемых признаков.* Слишком сложные модели с большим числом признаков могут быть склонными к переобучению, какие-то признаки могут быть не сильно важны в целом для решения, но использование их в модели вносит дополнительную ошибку, снижающую обобщающую способность и т.д. Например, для задачи предсказания стоимости дома признаки площади дома, количества комнат могут быть важны, но наличия детской площадки неподалеку – не сильно важны. Однако, решить какие признаки с модели оставить, а какие убрать – достаточно нетривиальная задача.
- 3) *Увеличить число признаков.* Возможно, что модель, наоборот, слишком проста, и ей не хватает дополнительных признаков, чтобы хорошо работать на новых данных. Например, для задачи предсказания стоимости дома можно добавить признаки наличия рядом магазина, материала, из которого построен дом и т.д. Проблемы добавления новых признаков в модель аналогичны проблемам увеличения набора данных – новые признаки нужно где-то взять (измерения, опросы, статистика, а это все требует времени), к тому же нужно решить, какие признаки стоит добавить.
- 4) *Добавление полиномиальных признаков.* В некоторых случаях добавление в модель различных степеней (второй, третьей и т.д.), а также других функций (тригонометрических, логарифмических и т.д.) от уже используемых признаков может улучшить модель. Примеры таких моделей мы рассматривали в разделе 2.3. Однако конструирование модели требует творческого подхода, экспериментов и интуиции.
- 5) *Настройка регуляризации.* Как мы обсудили в главе 4, регуляризация является действенным способом при проблемах с переобучением моделей. Можно увеличить параметр регуляризации, если модель слишком сложна и переобучается, и наоборот, уменьшить его, если модель недообучена. Важно только правильно отличать, когда модель переобучена, а когда недообучена. В некоторых случаях регуляризация может не помочь.

Таким образом, способов улучшения обобщающей способности достаточно много, но, во-первых, не все они могут помочь, а во-вторых, некоторые из них могут потребовать очень много времени (часы, дни и даже месяцы). Было бы полезно знать, какие из этих подходов и в каких ситуациях следует применять.

Для этого нам поможет *диагностика машинного обучения*. Это набор методик, которые позволяют оценить обобщающую способность модели, выявить недостатки модели и определить, какие из методов могут помочь в улучшении модели, а какие бесперспективны.

6.2. Оценивание обобщающей способности

Предположим, мы обучили три гипотезы регрессионной модели с разными степенями полинома на одном и том же наборе данных:

$$h_{\theta}^{(1)}(x) = \theta_0 + \theta_1 x;$$

$$h_{\theta}^{(2)}(x) = \theta_0 + \theta_1 x + \theta_2 x^2;$$

$$h_{\theta}^{(3)}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_4 x^4.$$

На рис. 6.1 показаны графики этих моделей: $h_{\theta}^{(1)}(x)$ на рис. 6.1, а, $h_{\theta}^{(2)}(x)$ на рис. 6.1, б и $h_{\theta}^{(3)}(x)$ на рис. 6.1, в. Точки обучающей выборки показаны красным цветом.

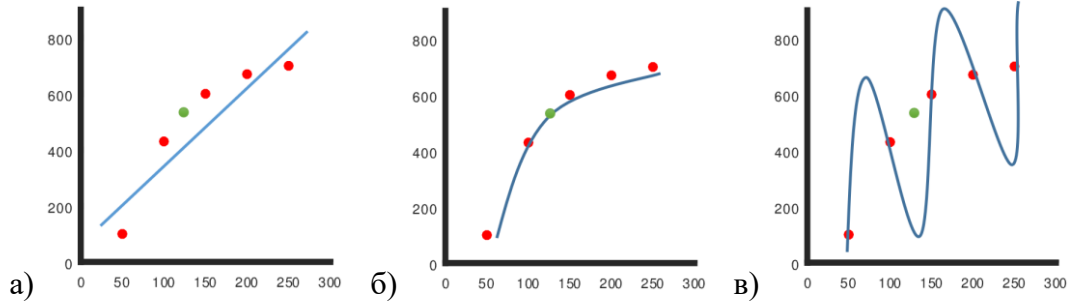


Рис. 6.1

Как мы разбирали в главе 4, $h_{\theta}^{(1)}(x)$ является недообученной моделью, $h_{\theta}^{(3)}(x)$, наоборот, переобученной, и только $h_{\theta}^{(2)}(x)$ можно считать имеющей неплохую обобщающую способность. Если каждой из этих моделей дать новые данные (зеленая точка на графике), то очевидно, что $h_{\theta}^{(1)}(x)$ и $h_{\theta}^{(3)}(x)$ дадут нам большую ошибку. Однако, если сравнить значения функции стоимости на обучающем наборе данных каждой из этих гипотез, то получится, что:

$$J^{(1)}(\theta) > J^{(2)}(\theta) > J^{(3)}(\theta).$$

То есть, модель с самой минимальной ошибкой на обучающей выборке может иметь самую низкую обобщающую способность.

Происходит это ввиду того, что, обучая модель, мы «подгоняем» параметры (вектор θ) под обучающую выборку. Мы получаем самый лучший вектор θ для этой выборки. Следовательно, оценить обобщающую способность модели на этой же выборке *невозможно*. Нам нужны новые данные, чтобы понять, насколько хорошо будет обобщаться модель.

В этом случае поступают следующим образом. Имеющиеся данные делят на две части: обучающая выборка (примерно 70%) и тестовая выборка (остальные 30%).

Обозначим обучающую выборку, как и раньше: $(x^{(i)}, y^{(i)})$, $i = 1, \dots, m$, m – объем обучающей выборки. Тестовую выборку обозначим: $(x_{test}^{(i)}, y_{test}^{(i)})$, $i = 1, \dots, m_{test}$, m_{test} – объем тестовой выборки.

Как и прежде, модель обучается с использованием обучающей выборки, минимизируя функцию стоимости $J(\theta)$ (возможно, с регуляризацией) и получая оптимальный вектор θ . Затем с использованием этого θ (и не меняя его) по тестовой выборке выполняется расчет тестовой функции стоимости (называют *ошибкой на тесте*, в отличие от функции стоимости без регуляризации на обучающей выборке, которую называют *ошибкой на обучении* $J_{train}(\theta)$, от $J(\theta)$ она отличается тем, что в $J(\theta)$ возможна регуляризация).

Для модели регрессии ошибка на обучении будет вычисляться как:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2.$$

Для задачи классификации можно определить ошибку на обучении как:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m err(h_{\theta}(x^{(i)}) - y^{(i)}) ,$$

где $err(h_{\theta}(x) - y)$ ошибка классификации, вычисляемая как:

$$err(h_{\theta}(x) - y) = \begin{cases} 1, & \text{если } (h_{\theta}(x) \geq 0.5 \text{ и } y = 0) \text{ или } (h_{\theta}(x) < 0.5 \text{ и } y = 1); \\ 0, & \text{в противном случае.} \end{cases}$$

То есть, выполняется подсчет количества ошибок классификации моделью (указаний неверного класса) и делится на объем выборки. Такой метод подходит для любого метода классификации: для логистической регрессии, для нейронной сети и других.

Для модели регрессии ошибка на тесте будет вычисляться как:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2 .$$

Для задачи классификации можно определить ошибку на тесте как:

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} err(h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)}) .$$

Значение $J_{test}(\theta)$ дает нам гораздо более обоснованную оценку обобщающей способности модели вследствие того, что выполняется по независимым от обучения данным.

Важно учесть, что разбиение начальной выборки на обучающую и тестовую часть *должно быть случайным*, то есть выборка перед разбиением перемешивается. Часто данные хранятся упорядоченными по какому-либо значению (по времени, по цене, по классам и т.д.). Разбивая не перемешанную выборку, мы очень рискуем получить неоднородные данные для обучения или тестирования.

6.3. Выбор модели обучения

Рассмотрим еще раз задачу линейной регрессии. Как показано в разделе 2.3, мы можем использовать различные полиномиальные модели для улучшения качества решения. Но какую степень полинома выбрать?

Пусть имеется десять моделей со степенями от 1 до 10:

$$\begin{aligned} h_{\theta}(x) &= \theta_0 + \theta_1 x; \\ h_{\theta}(x) &= \theta_0 + \theta_1 x + \theta_2 x^2; \\ &\vdots \\ h_{\theta}(x) &= \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10}. \end{aligned}$$

Добавим еще один параметр: d – степень полинома модели. Каждую из десяти моделей мы можем обучить на обучающем наборе и получить вектора параметров $\theta^{(1)}, \dots, \theta^{(10)}$ для каждой из моделей. Каждой модели будет соответствовать значение ошибки на обучении $J(\theta^{(i)})$, $i = 1, \dots, 10$. Но сравнивать между собой $J(\theta^{(i)})$ и по их значению решать, какая степень лучше, нельзя. Как показано в прошлом разделе, мы не можем оценивать качество модели по ошибке на обучении. Как правило, $J(\theta^{(1)}) > J(\theta^{(2)}) > \dots > J(\theta^{(10)})$, то есть наименьшее значение будет у модели самой высокой степени, и у нее же самая высокая вероятность переобучиться и получить плохую обобщающую способность.

В этой ситуации нам тоже поможет тестовая выборка. Мы можем получить ошибки на тесте $J_{test}(\theta^{(1)}), \dots, J_{test}(\theta^{(10)})$, и вот их сравнение уже позволяет обоснованно выбирать d , поскольку выполняется на независимых данных. Оптимальная степень определяется по минимальной ошибке на тесте (т.е. это степень той модели, которая меньше всего ошибается):

$$d^* = \min_d J_{test}(\theta^{(d)}).$$

Например, это модель с $d = 5$, полином пятой степени. Насколько хороша обобщающая способность этой модели? Можем ли мы ее оценить по $J_{test}(\theta^{(5)})$, если будем сравнивать с другими решениями: регрессиями на других функциях, нейронными сетями и т.д.

Как уже обсудили в прошлом разделе, не можем. Параметр d был подобран по тестовой выборке, и это значение оптимально для этой тестовой выборки. Тестовая выборка для параметра d превратилась в обучающую выборку. Это значит, что нам нужны еще отдельные данные для оценивания обобщающей способности полученного решения.

Таким образом мы подошли к другому разбиению обучающей выборки, которое используется в большинстве методов машинного обучения (модели с параметрами, регуляризацией, нейронные сети и т.д.).

Обучающую выборку делят на три части. Первая и самая большая часть, обычно около 60%, составляет *обучающая выборка*. На ней выполняется обучение моделей (получение вектора θ). Вторая часть называется *валидационной выборкой* и составляет примерно 20%. На ней выполняется подбор параметров (таких как d в примере выше). Наконец, третья часть называется *тестовой выборкой*, тоже обычно около 20%, и служит единственно для оценивания обобщающей способности полученного решения.

Как и ранее, обозначим обучающую выборку: $(x^{(i)}, y^{(i)})$, $i = 1, \dots, m$, m – объем обучающей выборки. Валидационную выборку будем обозначать: $(x_{cv}^{(i)}, y_{cv}^{(i)})$, $i = 1, \dots, m_{cv}$, m_{cv} – объем тестовой выборки (CV – сокращение от «cross validation», перекрестная проверка – это один из самых распространенных методов валидации, но их мы будем обсуждать позднее). Тестовую выборку обозначим, как прежде: $(x_{test}^{(i)}, y_{test}^{(i)})$, $i = 1, \dots, m_{test}$, m_{test} – объем тестовой выборки.

Ошибку, полученную на валидационном наборе:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

будем называть *ошибкой на валидации* или *ошибкой валидации*.

6.4. Дилемма смещения/дисперсии

Рассмотрим снова пример из предыдущего раздела с полиномиальными регрессионными моделями различных степеней d и посмотрим, как изменяются значения ошибки на обучении и на валидации в зависимости от d . Рис. 6.2. иллюстрирует этот процесс.

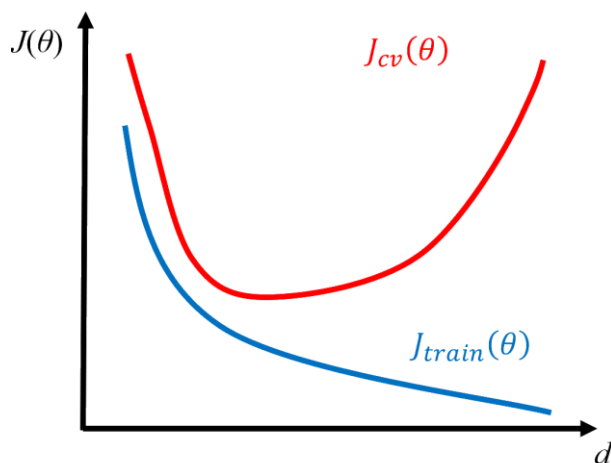


Рис. 6.2

Сначала рассмотрим ошибку на обучении $J_{train}(\theta)$ (синий график). Модель низкой степени (первая, вторая) для большинства данных будет недообученной (см. рис. 6.1, а). Это приведет к тому, что ошибка в левой части графика 6.2 будет большой. Но с повышением степени полинома, функция становится все более сложной и лучше приспособливается к точкам обучающей выборки, ошибка падает. Когда степень полинома приближается к числу точек выборки, $J_{train}(\theta)$ может дойти до нуля (полином 99 степени может с нулевой

ошибкой пройти через 100 точек). На графике (рис. 6.2) в правой части функция $J_{train}(\theta)$ стремится к нулю.

Рассмотрим теперь поведение функции ошибки на валидации $J_{cv}(\theta)$, которая вычисляется на других данных, не участвующих в обучении (красный график). На низких значениях степени полинома обобщающая способность модели низка (недообученная модель), и значение $J_{cv}(\theta)$ будет велико. Как правило, больше $J_{train}(\theta)$, поскольку $J_{train}(\theta)$ вычисляется на обучающей выборке и θ подогнано к ее минимуму, а валидационная выборка произвольна.

Повышение степени полинома приводит к тому, что обобщающая способность возрастает, $J_{cv}(\theta)$ становится меньше. Но это снижение не может быть постоянным, как в случае с $J_{train}(\theta)$. В определенный момент степень полинома станет такой большой, что функция начнет переобучаться (например, рис. 6.1., в), обобщающая способность падать, и $J_{cv}(\theta)$ будет увеличиваться.

Ситуация, когда $J_{train}(\theta)$ и $J_{cv}(\theta)$ велики (левая часть графика на рис. 6.2), сигнализирует о том, что модель недообучена, слишком грубая либо неверно выбраны функции гипотезы. Получить хорошее решение на такой гипотезе невозможно. Ситуацию называют *проблемой смещения* (англ. bias – базис, основа, точка отсчета). Можно понимать, что модель «смещена» от правильного решения, и как параметры не подбери, хорошего решения на такой модели не найти.

Ситуация в правой части графика означает, что модель переобучена, имеет слишком много параметров, большую сложность и вариабельность. Такую ситуацию называют *проблемой дисперсии* или *проблемой вариабельности* (англ. variance – вариабельность, дисперсия). Решения, которые можно получить на такой модели подбором параметров, могут быть самыми разными от очень хороших до очень плохих. Проблема, однако в том, что вероятность наткнуться случайно на хорошее решение очень мала.

Дилемма смещение/дисперсия (англ. bias/variance) заключается в том, что, например, на одном конце находятся решения слишком простые, а на другом слишком сложные. И те, и другие не обладают хорошей обобщающей способностью (см. кривую $J_{cv}(\theta)$), а хорошие решения лежат где-то посередине.

6.5. Подбор параметра регуляризации

Рассмотрим задачу регуляризованной линейной регрессии с полиномиальной моделью:

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4.$$

Функция стоимости для регуляризованной регрессии определяется как:

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right].$$

От параметра регуляризации λ , как было отмечено в разделе 4.3, зависит влияние регуляризации на решение и, как следствие, обобщающая способность модели. Выбор правильного значения для λ важная задача.

В этой задаче нам может также помочь разбиение выборки на обучающую и валидационную, как это было в разделе 6.3.

Можно взять различные значения λ , например из множества (обычно берется экспоненциальный ряд или просто множество с возрастающим шагом):

$$\Lambda = \{0, 0.01, 0.02, 0.04, 0.08, 0.16, \dots, 10\}.$$

Можно обучить модель с каждым значением λ и получить оптимальные вектора θ :

$$\begin{aligned} \theta^{(1)} &= \min_{\theta} J(\theta | \lambda = 0); \\ \theta^{(2)} &= \min_{\theta} J(\theta | \lambda = 0.01); \\ \theta^{(3)} &= \min_{\theta} J(\theta | \lambda = 0.02); \end{aligned}$$

$$\vdots$$

$$\theta^{(12)} = \min_{\theta} J(\theta | \lambda = 10).$$

Аналогично, как мы подбирали наилучшее значение для показателя степени d , наилучшее значение для λ можно найти, посчитав валидационную ошибку для каждого вектора $\theta^{(i)}$. Лучшее значение λ будет соответствовать той модели, которая имеет лучшую обобщающую способность по результатам обучения, то есть наименьшую ошибку валидации:

$$\lambda^* = \Lambda(k), \quad k = \min_i J_{cv}(\theta^{(i)}).$$

В результате мы получим модель с обоснованно выбранным значением параметра регуляризации. Оценить же обобщающую способность этой модели можно с привлечением уже тестовой выборки, как $J_{test}(\theta^{(k)})$.

Примечание: не стоит только забывать, что ошибки $J_{train}(\theta)$, $J_{cv}(\theta)$, $J_{test}(\theta)$ вычисляются без регуляризации, в отличие от функции стоимости $J(\theta)$.

Можно также рассмотреть, как ведут себя кривые ошибки на обучении и на валидации в зависимости от значения параметра λ . Соответствующие кривые показаны на рис. 6.3.

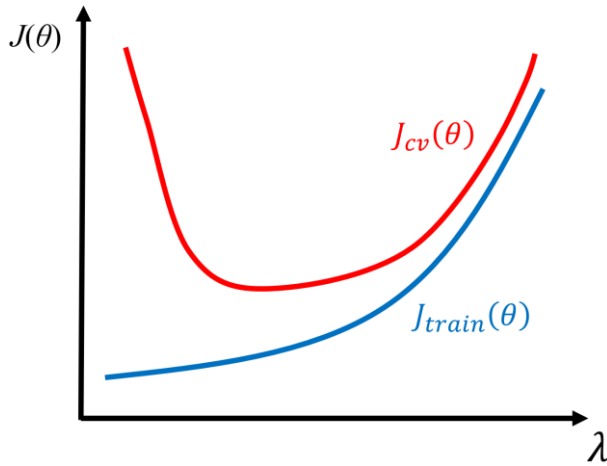


Рис. 6.3

Когда значение λ мало, регуляризация почти не оказывает влияния на решение. Значение ошибки на обучении $J_{train}(\theta)$ практически равно функции стоимости $J(\theta)$. Но при увеличении значения λ величина $J_{train}(\theta)$ будет возрастать, поскольку θ подобран для $J(\theta)$ с регуляризацией и хуже подходит для $J_{train}(\theta)$ без регуляризации.

Для графика $J_{cv}(\theta)$ мы будем наблюдать вид, очень похожий на рис. 6.2. Большое расхождение между $J_{train}(\theta)$ и $J_{cv}(\theta)$, высокое значение $J_{cv}(\theta)$ свидетельствуют о наличии проблемы дисперсии (левая часть графика), в то время как близкие и одновременно большие значения $J_{train}(\theta)$ и $J_{cv}(\theta)$ свидетельствуют о наличии проблемы смещения. Хорошие решения лежат в диапазоне тех значений λ , где величина $J_{cv}(\theta)$ минимальна.

6.6. Кривые обучения

Построение графиков зависимости $J_{train}(\theta)$ и $J_{cv}(\theta)$ от степени полинома или параметра регуляризации позволяет оценить наличие проблемы в обучаемой модели. Но что, если нужно оценить модели с различными d и λ в совокупности? Будет необходимо построить и посмотреть на вид трехмерных поверхностей $J_{train}(\theta)$ и $J_{cv}(\theta)$? А если параметров будет больше? Например, выбор количества слоев в нейронной сети, количества нейронов в каждом слое и параметра регуляризации. Визуально отобразить и проанализировать такой график будет невозможно.

В данном случае нам поможет более универсальный инструмент, который называют *кривыми обучения*. Это все те же графики функций $J_{train}(\theta)$ и $J_{cv}(\theta)$, но только от параметра

m – размера обучающей выборки. Кривые обучения не так наглядны, как графики от параметров на рис. 6.2 и рис. 6.3, но обладают большим достоинством: они всегда одномерные кривые на плоскости, сколько бы параметров не было в обучаемой модели. Значит из можно всегда проанализировать.

Чтобы построить кривые обучения, прибегают к следующему трюку: сначала берут из обучающей выборки одну случайную точку, обучают модель на выборке из одной этой точки, вычисляют значение $J_{train}(\theta)$ и $J_{cv}(\theta)$. Затем добавляют еще одну точку, потом еще одну. Весь процесс повторяется, пока в обучающей выборке не окажутся все m элементов.

Обратите внимание, что разбиваем на части мы только обучающую выборку. Валидационная выборка остается не тронутой, и ошибка валидации всегда вычисляется по всем m_{cv} значениям.

Возьмем для примера регрессионную модель:

$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2.$$

Рассмотрим, как выглядят кривые обучения (рис. 6.4). Мы начинаем со значения $m = 1$, т. е. с обучающей выборки в один элемент. На этой обучающей выборке мы обучаем модель и получаем вектор θ . Можно всегда можно подобрать значения θ таким образом, что ошибка на обучении будет минимальна. (Если используется модель без регуляризации, то ошибка будет попросту нулевой. Для модели с регуляризацией некоторое значение ошибки будет из-за влияния регуляризации.) На рис. 6.5, а показана такая ситуация. Таким образом, значение $J_{train}(\theta)$ (синяя кривая) будет очень малым (левая часть графика).

Что при этом с функцией $J_{cv}(\theta)$ (красная кривая)? Во-первых, нужно не забывать, что, в то время как $J_{train}(\theta)$ строится по 1,2,3 и т.д. до m точек, $J_{cv}(\theta)$ на каждом этом шаге всегда вычисляется по своим отдельным m_{cv} точкам. На первом шаге ($m = 1$) мы получим модель с очень плохой обобщающей способностью, значение $J_{cv}(\theta)$ будет большим.

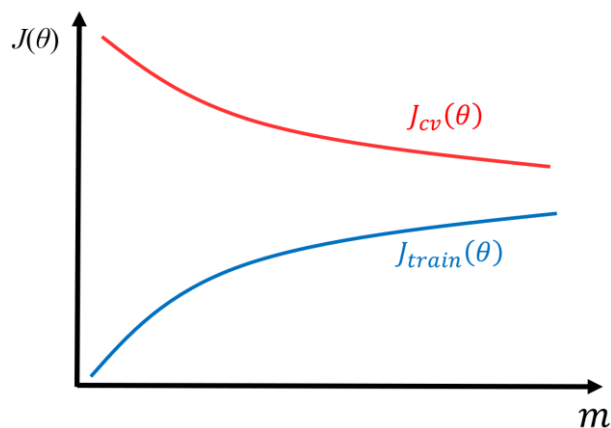


Рис. 6.4

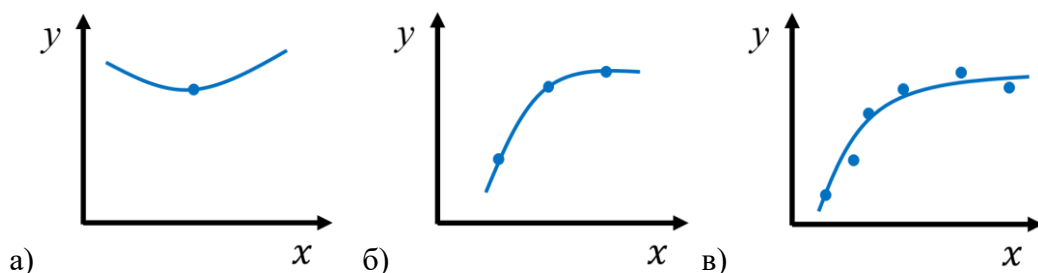


Рис. 6.5

Пока точек не много (рис. 6.5, б) значение ошибки на обучении все еще будет не велико, однако, когда в обучающую выборку попадает все больше и больше точек (рис. 6.5, в) ошибка на обучении будет возрастать. Ошибка валидации, напротив, будет уменьшаться, поскольку чем больше точек, тем более адекватное решения мы получим, с

большой обобщающей способностью.

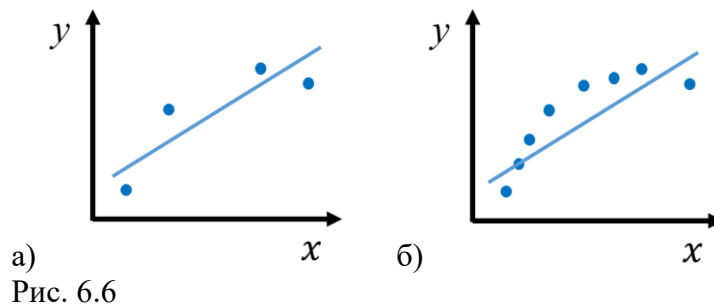
Рассмотрим теперь, как будут выглядеть эти графики при наличии у модели проблем смещения или дисперсии, чтобы понять, как их можно диагностировать.

Пусть наша гипотеза имеет вид:

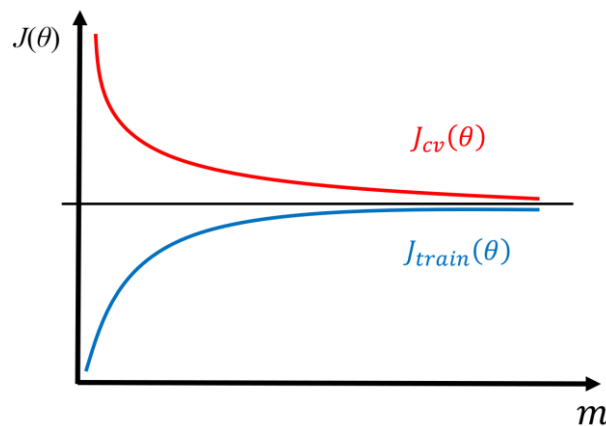
$$h_{\theta}(x) = \theta_0 + \theta_1 x.$$

Это очень простая гипотеза, и, если данные не подчиняются линейному закону, мы наверняка получим проблему смещения.

При малых значениях m мы получим некоторый вид нашей гипотезы, как, например, показано на рис. 6.6, а. При увеличении числа точек (рис. 6.6, б), сильно не поменяется. Новым точкам больше нечего уточнить для такой простой модели.



Как при этом будут выглядеть кривые обучения, иллюстрирует рис. 6.7. Ошибка на обучении быстро возрастает на первых же точках, затем устанавливается и практически прекращает свой рост. Ошибка валидации на первых точках наоборот, быстро снижается, но тоже до определенной степени, а затем тоже практически перестает уменьшаться. С добавлением новых точек обобщающая способность модели не улучшается. Обе кривые располагаются близко друг от друга, как бы приближаясь к некоей разделяющей границе.



Таким образом, если мы видим ситуацию, похожую на рис. 6.7, то очень вероятно, что имеется проблема смещения. Увеличение обучающей выборки не поможет (кривые не изменятся), помочь может усложнение модели.

Рассмотрим теперь, как понять, что модель имеет проблему дисперсии. Пусть гипотеза имеет вид:

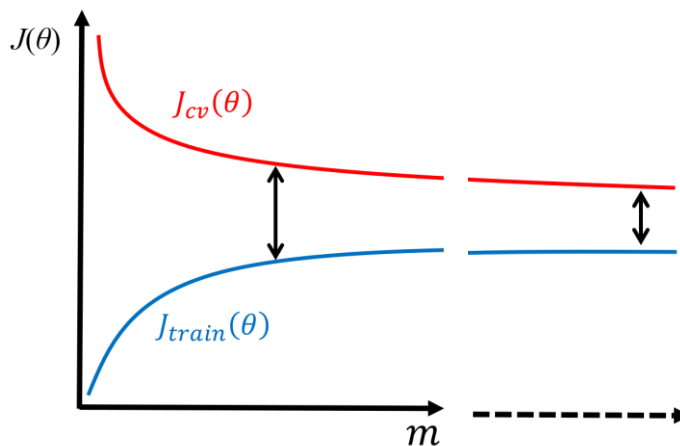
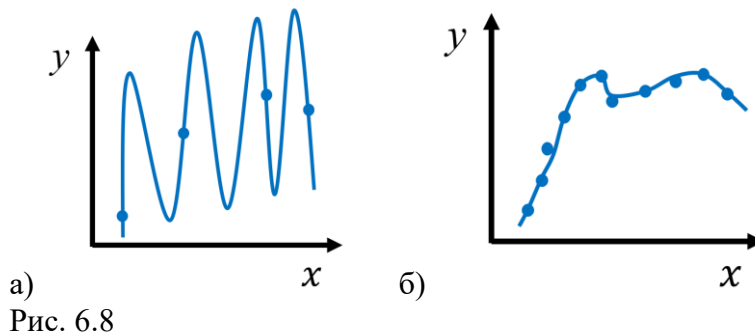
$$h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_{100} x^{100}.$$

Как мы уже многократно рассматривали, гипотезы такого сложного вида склонны к переобучению (соответственно имеют проблему дисперсии).

Когда точек обучающей выборки мало (для малых m), ошибка $J_{train}(\theta)$ будет очень мала, потому что сложная функция гипотезы будет адаптироваться к точкам (рис. 6.8, а). С

увеличением числа точек ошибка на обучении будет расти. Функция гипотезы будет адаптироваться к новым точкам (рис. 6.8, б), но добавление каждой новой точки в выборку будет немного повышать величину ошибки. Таким образом, график функции $J_{train}(\theta)$ (синяя линия на рис. 6.9) будет плавно увеличиваться.

Ошибка $J_{cv}(\theta)$ (красная линия на рис. 6.9) при малых m имеет самое высокое значение (обученная на малой обучающей выборке модель имеет самую плохую обобщающую способность), но с увеличением числа точек плавно снижается.



Особенностью модели с проблемой дисперсии является то, что между кривыми $J_{train}(\theta)$ и $J_{cv}(\theta)$ наблюдается заметный разрыв (на рис. 6.9 показан вертикальной линией со стрелками), и при этом кривая $J_{train}(\theta)$ имеет тенденцию к росту, а кривая $J_{cv}(\theta)$ – тенденцию к уменьшению. Они не достигли своего предела, как на графике рис. 6.7. Если еще больше увеличить m (то есть найти новые данные и добавить их к обучающей выборке, на рис. 6.9 показано пунктирной линией оси), то можно улучшить модель и уменьшить ошибку валидации.

Если кривая $J_{cv}(\theta)$ в процессе обучения после какого-то m перестает уменьшаться, значит дальнейшее обучение не целесообразно, больший объем выборки не улучшит решение.

6.7. Диагностика и выбор метода повышения качества модели

Вернемся к исходному вопросу, с которого мы начали эту главу. Мы построили модель регрессии, нашли оптимальный вектор θ , но модель имеет плохую обобщающую способность (результаты, которые выдает наша гипотеза на новых данных, на которых она не обучалась, плохо соответствуют реальности).

В этой ситуации, мы можем построить кривые обучения и посмотреть на их вид. Если мы видим ситуацию, похожую на рис. 6.7, то очень вероятно, что наша модель имеет проблему смещения. Если графики похожи на рис. 6.9, то вероятно, что в наличии проблема

дисперсии. В разделе 6.1 перечислены методы, которые могут помочь в повышении обобщающей способности. Применять их бездумно все подряд – плохое решение, некоторые из них могут потребовать больших усилий, но не принести результата.

Если наша модель страдает от проблемы смещения, то разумно попробовать следующее:

1) *Уменьшить параметр регуляризации λ* , если мы используем регуляризацию. Как показано в разделе 6.5, слишком большое значение λ может привести к проблеме смещения.

2) *Добавить полиномиальные признаки*. Если гипотеза слишком проста и не в состоянии адаптироваться под сложную нелинейную зависимость в обучающей выборке, то различные полиномиальные степени признаков (как показано в разделе 2.3) могут помочь.

3) *Увеличить число используемых признаков*. Возможно, что признаков, описывающих объекты, не достаточно для решения задачи. Тогда их нужно найти и дополнить описание объектов новыми данными. (Например, для задачи медицинской диагностики добавить к описанию пациента новые анализы и симптомы.) Это самый трудозатратный способ, поскольку может потребовать новых измерений, опросов и т.д. К нему стоит прибегать, если предыдущие не помогают.

Другие способы повышения обобщающей способности при наличии проблемы смещения не помогут: будут бесполезны или даже усугубят ее. Но они могут помочь при наличии проблемы дисперсии:

1) *Увеличить параметр регуляризации λ* . Самое простое, что можно попробовать при дисперсии, это увеличить λ . Или добавить в модель регуляризацию, если ее нет.

2) *Уменьшить число используемых признаков*. Если модель слишком сложна, то можно уменьшить число используемых признаков, убрать из модели малозначимые признаки, убрать некоторые полиномиальные признаки. Регуляризация выполняет похожую задачу, но не разбирается, какой признак важнее, а какой нет. Возможно, что обоснованный выбор признаков улучшит решение.

3) *Увеличить обучающий набор данных*. Если другие подходы не помогают, то остается искать новые данные для обучения и увеличивать объем выборки. По виду кривой ошибки валидации от размера обучающей выборки можно понять, помогут ли улучшить решение новые данные: если уменьшение ошибки валидации не перестало уменьшаться после какого-то значения m , то добавление новых данных все еще может улучшить обобщающую способность.

6.8. Рекомендации по разработке моделей машинного обучения

При разработке модели машинного обучения, какая бы не стояла задача – классификации или регрессии – сложно сразу определить, каким методом ее решать, как выбрать признаки и т.д. Решение выбранное интуитивно зачастую может оказаться не пригодным, поэтому желательно, чтобы выбор был обоснованным.

В самом начале рекомендуется не браться за разработку сразу сложной модели, а попробовать самое простое решение, которое можно быстро реализовать. Обучить модель, построить кривые обучения, чтобы понять, что стоит делать дальше.

Валидационный набор данных можно использовать для обоснованного подбора признаков для модели. Подготовив различные варианты признаков, можно сравнить ошибку на валидации и выбрать вариант, который подходит лучше.

Тестовый набор позволит оценить обобщающую способность модели. Также его можно использовать для сравнения моделей между собой (если подготовлено несколько вариантов).

6.9. Несимметричные классы

Рассмотрим задачу классификации на два класса ($y = 0$ и $y = 1$), в которой один класс ($y = 1$) встречается гораздо реже, чем другой ($y = 0$), т.е.

$$\#[y = 1] \ll \#[y = 0].$$

Такие классы и такие задачи называют несимметричными, и они имеют ряд особенностей в оценивании качества решения этих задач методами машинного обучения.

Обычно, если необходимо оценить качество классификации, прибегают к такому простому критерию, как мера достоверности (*accuracy*):

$$A = \frac{T}{m_{test}},$$

где T – количество правильных предсказаний модели (т.е. количество объектов тестовой выборки, для которых $h_{\theta}(x^{(i)}) = y^{(i)}$), m_{test} – объем тестовой выборки. Вместо тестовой выборки может быть валидационная. Чем больше достоверность, тем больше правильных ответов дает модель.

Пример: задача медицинской диагностики ракового заболевания: $y = 0$ означает, что у пациент не обнаружен рак, и $y = 1$ означает, что заболевание выявлено. Допустим, что была создана и обучена модель, которая показала достоверность 99% ($A = 0.99$). Это достаточно хороший результат, если бы речь шла про обычную классификацию. Но предположим, что в реальности только 0.5% пациентов действительно имеют раковое заболевание. Тогда результат $A = 0.99$ уже не является таким хорошим.

Возьмем гипотезу вида:

$$h_{\theta}(x) = 0,$$

то есть мы всегда выдаем ответ, что раковое заболевание не обнаружено для любого входного x . В таком случае достоверность этой гипотезы будет даже еще лучше: 99,5% ($A = 0.995$), ведь в тестовой выборке только 0.5% пациентов с раком. ■

То, что такая совершенно неправильная гипотеза имеет высокую достоверность, свидетельствует о том, что эта мера плохо подходит для оценки несимметричных классов. Если на основе достоверности пытаться оценить, какие признаки лучше подходят для решения задачи или какую модель выбрать, нельзя точно понять, внесенные в модель изменения действительно приводят к улучшению решения, или наоборот, повышение достоверности является результатом ошибки, подобному примеру выше.

6.10. Метрики точности и полноты

Для более адекватной оценки моделей, в том числе и с несимметричными классами, используют другие *метрики* качества моделей. Будем в будущем считать, что редким классом в несимметричной задаче является $y = 1$.

Рассмотрим четыре класса результатов, которые может выдать модель для каждого из тестовых объектов:

- 1) Если модель выдает результат $h_{\theta}(x^{(i)}) = 1$ и для этого объекта $y^{(i)} = 1$, такие ответы называют *истинно положительными* (true positive), что означает, что модель обнаружила искомый класс на объекте (например, диагноз, как примере выше) и выдала правильный результат.
- 2) Если модель выдает результат $h_{\theta}(x^{(i)}) = 0$ и для этого объекта $y^{(i)} = 0$, такие ответы называют *истинно отрицательными* (true negative), что означает, что модель не обнаружила искомый класс на объекте и выдала правильный результат.
- 3) Если модель выдает результат $h_{\theta}(x^{(i)}) = 1$, но для этого объекта $y^{(i)} = 0$, такие ответы называют *ложно положительными* (false positive), что означает, что модель обнаружила искомый класс на объекте, но на самом деле его не должно быть, т.е. выдала неправильный результат.
- 4) Если модель выдает результат $h_{\theta}(x^{(i)}) = 0$, но для этого объекта $y^{(i)} = 1$, такие ответы называют *ложно отрицательными* (false negative).

Запустив модель на тестовой (или валидационной) выборке можно подсчитать коли-

чество ответов каждого класса. Обозначим: TP – количество истинно положительных исходов, TN – количество истинно отрицательных исходов, FP – количество ложно положительных исходов, FN – количество ложно отрицательных исходов.

Можно составить матрицу (рис. 7.1) размером 2×2 , в которой столбцам будут соответствовать актуальные значения (из выборки), а строкам – результаты работы модели. Количество каждого из исходов помещается в соответствующие ячейки матрицы: истинно положительные на пересечение столбца ($y = 1$) и строки ($h_\theta(x) = 1$) и т.д.

		Актуальные	
		$y = 1$	$y = 0$
Предсказанные	$h(x) = 1$	TP	FP
	$h(x) = 0$	FN	TN

Рис. 7.1

Поскольку каждый результат относится к какому-либо из четырех классов, их сумма будет равна объему выборки. Количество истинных ответов ($TP + TN$) соответствует общему числу правильных предсказаний модели, а количество ложных ответов ($FP + FN$) соответствует числу ошибок модели. Но теперь правильные и ошибочные ответы разделены и известно сколько правильных и ошибочных ответов по каждому классу отдельно.

Зная такую матрицу, можно посчитать достоверность модели:

$$A = \frac{TP + TN}{TP + FP + TN + FN}.$$

Как уже было показано, достоверность неинформативна для несимметричных классов. Рассмотрим еще две метрики.

Точностью (precision) называют метрику, вычисляемую как:

$$P = \frac{TP}{TP + FP}.$$

Точность показывает нам, какая часть среди всех ответов $h(x) = 1$ совпадает с актуальными $y = 1$ (исходы с $h(x) = 0$ здесь не рассматриваются).

Для рассмотренного выше примера задачи медицинской диагностики рака, точность показывает, какая часть из всех пациентов, кому модель предсказала рак, действительно им больна.

Полнотой (recall) называют метрику, вычисляемую как:

$$R = \frac{TP}{TP + FN}.$$

Полнота показывает, какая часть среди всех актуальных $y = 1$, совпадает с ответами $h(x) = 1$ (варианты с $y = 0$ не рассматриваются).

Для задачи медицинской диагностики рака, полнота показывает, какой части из всех пациентов, больных раком, модель правильно поставила диагноз.

Использование этих двух метрик для несимметричных классов позволит выявить действительно правильно работающие модели. Например, для модели с гипотезой $h_\theta(x) = 0$ (т.е. всегда предсказывающей отсутствие рака), $R = 0$ и $P = 0$.

6.11. Компромисс между точностью и полнотой

Предположим, задача классификации решается методом логистической регрессии. Как показано в главе 3, гипотеза выдает результат в интервале: $0 \leq h(x) \leq 1$. Этот результат мы

интерпретируем как вероятность отнесения объекта x к классу 1. В главе 3 мы использовали следующие условия:

если $h(x) \geq 0.5$, то предсказываем класс 1;
 если $h(x) < 0.5$, то предсказываем класс 0.

Предположим, что мы хотим идентифицировать класс 1 только в том случае, если очень уверены в этом. Тогда мы можем поднять порог в условии и использовать вместо 0.5, например 0.7. В этом случае мы будем избегать ложно положительного исхода.

Если же цель у нас обратная – пропустить как можно меньше реальных объектов класса 1 (избежать ложно отрицательных исходов), то порог можно уменьшить (например, 0.3).

То есть условие становится следующим:

если $h(x) \geq \tau$, то предсказываем класс 1;
 если $h(x) < \tau$, то предсказываем класс 0.

Если подсчитать различные исходы при различных параметрах порога τ в интервале $[0, 1]$, то для каждого из них будут свои значения точности и полноты. Можно построить график отношения между этими метриками (рис. 7.2, а).

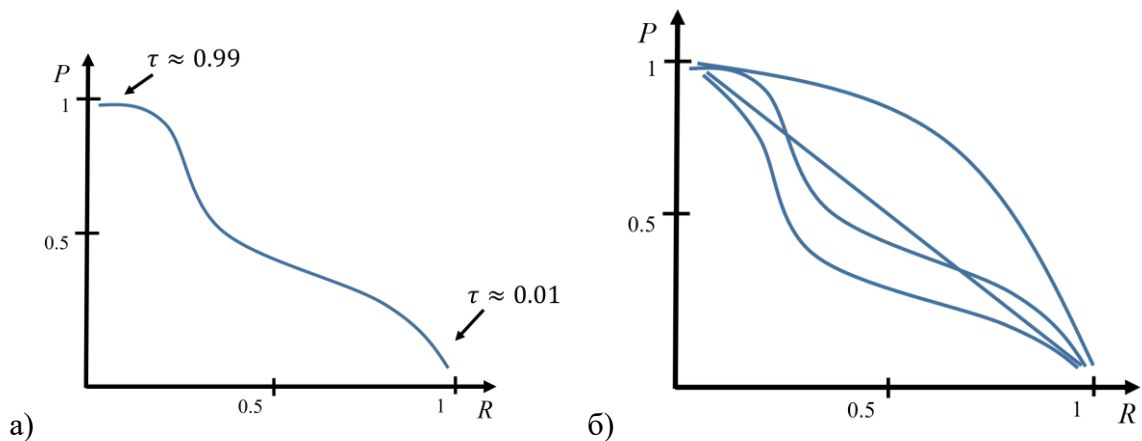


Рис. 7.2

Максимальному значению точности будет соответствовать наибольшее значение порога τ , а максимальному значению полноты – минимальное значение τ . Вид кривой может быть достаточно произвольным, как показано на рис. 7.2, б.

6.12. F-мера

Предположим, что имеется несимметричная задача классификации, решая которую необходимо подобрать вид модели, различные варианты признаков, регуляризацию и другие параметры, таким образом, чтобы минимизировать число ошибок модели. Как было показано в разделе 6.9, простой подсчет достоверности не даст нам адекватной оценки для несимметричных классов.

Пусть имеет три варианта решения, назовем их «Алгоритм 1», «Алгоритм 2» и «Алгоритм 3». Мы можем посчитать и оценить метрики точности и полноты для этих решений (см. табл. 7.1). Однако возникает вопрос, как сравнивать эти решения, имея два набора метрик: например, какое решение лучше, с показателями 0.4 и 0.5 или с показателями 0.7 и 0.1? Для сравнения желательно иметь один численный показатель, аккумулирующий в себя качество решения.

Табл. 7.1

Решение	Точность P	Полнота R	F -мера
Алгоритм 1	0.5	0.4	0.44
Алгоритм 2	0.7	0.1	0.17
Алгоритм 3	0.02	1.0	0.04

В математической статистике есть такая мера, которая называется F_β -мерой:

$$F_\beta = (1 + \beta^2) \frac{PR}{\beta^2 P + R}.$$

В зависимости от параметра β мера может иметь различные свойства. Нам достаточно меры $\beta = 1$, которая вычисляется по следующей простой формуле:

$$F_1 = 2 \frac{PR}{P + R}.$$

Мера F_1 обладает полезными для нас свойствами: она принимает значения в интервале $[0, 1]$; если $P = 0$ и $R = 0$, то $F_1 = 0$; если $P = 1$ и $R = 1$, то $F_1 = 1$; если P или R повышается, то F_1 тоже растет.

6.13. Использование больших данных

Большие данные (Big Data) – это подход в машинном обучении, когда для настройки алгоритма используются очень большие обучающие наборы (миллионы, миллиарды и более). Технологии больших данных стали очень популярны в XXI веке.

В 2001 году американскими учеными Бэнко и Бриллом была исследована задача классификации близких по звучанию слов английского языка с использованием нескольких различных алгоритмов классификации и большим набором данных.

Их исследование показало, что все алгоритмы классификации показывают примерно одинаковое качество предсказания, и что более важно, что с увеличением объема обучающей выборки качество только росло.

Многочисленные эксперименты других исследователей на других задачах тоже подтверждали этот вывод.

В разделах 6.4–6.6 мы рассмотрели дилемму смещения/дисперсии. Увеличение объема выборки, как известно, не всегда хорошо работает. Если задача имеет проблему смещения, то большой объем обучающих данных не должен привести к улучшению качества классификации.

Действительно, подход Big Data пригоден не для всех задач. Как оценить, когда использование больших данных оправдано?

Во-первых, вектор параметров x должен содержать достаточно информации для точного предсказания y . Оценить это не всегда просто, но есть один полезный интуитивный подход, который часто помогает: если человек-эксперт в предметной области задачи на основе информации x может дать уверенное предсказание y , то ее скорее всего достаточно.

В задаче Бэнко и Брилла необходимо было определить, какое слово в предложении является пропущенным. Например:

For breakfast I ate ____ eggs. (варианты: two, to, too)

Человек в подавляющем большинстве случаев безошибочно может определить, какое из вариантов слов подходит для этого предложения.

Другой пример: определение стоимости дома по его площади. В этой задаче данных явно недостаточно. Если опросить ряд риелторов с вопросом, сколько может стоить дом площадью 150 м^2 , от каждого из них будет масса уточняющих вопросов (из чего сделан, сколько этажей, где расположен и т.д.).

Во-вторых, для решения этой задачи подбирается алгоритм с большим числом параметров (логистическая регрессия с большим числом признаков, нейронная сети с множеством скрытых слоев и т.д.).

Таким образом мы получаем задачу, которая с большой вероятностью имеет низкое смещение и большую дисперсию.

В задачах с большой дисперсией значение ошибки на обучении $J_{\text{train}}(\theta)$ мало (вплоть до нуля, см. рис. 6.2). Но проблема большой дисперсии в том, что модель может переобучиться и иметь большую ошибку на валидации. Как показано в разделе 6.6, одним

из способов борьбы с переобучением является увеличение выборки. При достаточно большом объеме выборки значение ошибки на валидации или тесте приближается к значению ошибки на обучении и тоже становится мало. То есть мы получаем «хорошую» предсказывающую модель.

Таким образом, фокус эффективности больших данных в том, что мы берем сложную модель с большим числом параметров (такую, что с большой вероятностью бы переобучилась), но огромный набор данных для обучения (тысячи, миллионы и миллиарды примеров) возможность переобучения нивелирует.

Стоит иметь в виду, что это возможно только в том случае, если признаков описания объекта достаточно, чтобы сделать правильное предсказание. Большинство неудач в больших данных связано именно с этой проблемой.

Подход больших данных в современном машинном обучении как правило реализуется с использованием больших нейронных сетей с множеством скрытых слоев (нейронные сети глубокого обучения).

Вопросы и задания

1) Предположим, что модель линейной регрессии без регуляризации очень сильно переобучена. Какими будут значения функции стоимости на обучающей выборке ($J(\theta)$) и на тестовой выборке ($J_{test}(\theta)$)?

- а) $J(\theta)$ будет мало, а $J_{test}(\theta)$ будет велико
- б) $J(\theta)$ будет мало и $J_{test}(\theta)$ будет мало
- в) $J(\theta)$ будет велико, а $J_{test}(\theta)$ будет мало
- г) $J(\theta)$ будет велико и $J_{test}(\theta)$ будет велико

2) Вы построили полиномиальную регрессионную модель, выбрали степень полинома d , и оценили обобщающую способность полученной модели. При этом обнаружили, что значение ошибки $J_{cv}(\theta)$ меньше, чем $J_{test}(\theta)$. По какой причине это происходит?

- а) Потому что параметр d подобран по тестовой выборке
- б) Потому что параметр d подобран по валидационной выборке
- в) Потому что валидационная выборка обычно меньше тестовой
- г) Потому что валидационная выборка обычно больше тестовой

3) Вы решаете задачу классификации. В процессе обучения вы получили ошибку классификации на обучающем наборе, равную 0.1, и на валидационном наборе, равную 0.3. С какой проблемой вероятнее всего вы столкнулись?

- а) Смещения (переобучения)
- б) Смещения (недообучения)
- в) Дисперсии (переобучения)
- г) Дисперсии (недообучения)

4) Рассмотрим задачу линейной регрессии с регуляризацией. Функция стоимости определяется как:

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^n \theta_j^2 \right];$$

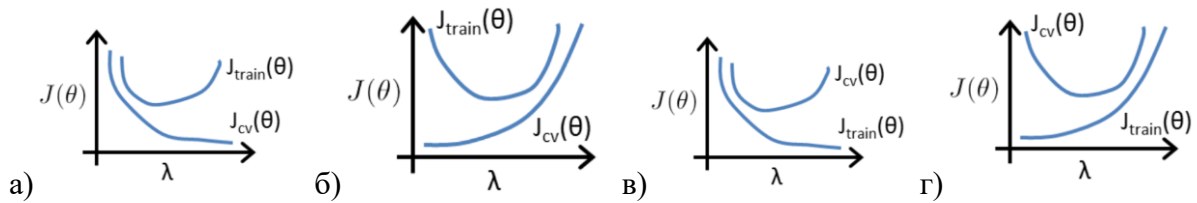
ошибка на обучении:

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2;$$

ошибка на валидации:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2.$$

Как будут выглядеть кривые зависимости $J_{train}(\theta)$ и $J_{cv}(\theta)$ от параметра λ ?



5) Вы обучаете нейронную сеть с одним скрытым слоем. В процессе обучения вы определили, что ошибка на валидационном наборе сильно больше, чем на обучающем наборе. Может ли вам помочь в этой ситуации увеличение числа скрытых слоев нейронной сети?

- а) Да, поскольку это увеличивает число параметров и позволяет представить более сложную функцию.
- б) Да, поскольку имеется проблема смещения.
- в) Нет, поскольку имеется проблема смещения.
- г) Нет, поскольку имеется проблема дисперсии.

6) Вы обнаружили, что обученная модель допускает много ошибок, и решили провести анализ ошибок и сравнить несколько различных вариантов решения. Почему это стоит делать на валидационном наборе данных, а не на тестовом?

- а) Валидационный набор обычно больше тестового.
- б) Ошибка на валидационном наборе обычно меньше.
- в) Если мы подберем вариант решения по тестовому набору, то не сможем оценить обобщающую способность модели.
- г) Использование валидационного набора даст меньший набор признаков.

7) Вы проверили модель на тестовом наборе данных и нашли, что количество истинно положительных ответов 80, ложно положительных 20, ложно отрицательных 80, истинно отрицательных 820. Какие значения принимают достоверность, точность и полнота?

8) Используя модель логистической регрессии для двух классов, мы предсказываем класс 1, если $h(x) \geq \tau$, и класс 0, если $h(x) < \tau$, где τ – некоторый порог. Каким способом подобрать оптимальные значения для порога τ ?

- а) Вычислить значения P и R на тестовом наборе данных и выбрать значение τ при котором достигается максимум любого из них.
- б) Вычислить значения P и R на валидационном наборе данных и выбрать значение τ при котором достигается максимум.
- в) Вычислить значения P и R на тестовом наборе данных и выбрать значение τ при котором достигается максимум F-меры.
- г) Вычислить значения P и R на валидационном наборе данных и выбрать значение τ при котором достигается максимум F-меры.

9) В каких ситуациях из перечисленных ниже использование больших данных вероятно не поможет решить задачу?

- а) Признаки x не содержат достаточно информации для достоверного предсказания.
- б) Используется алгоритм с большим числом параметров.
- в) Признаки x не содержат достаточно информации для достоверного предсказания, но используется алгоритм с большим числом параметров.
- г) Не используется регуляризация (или параметр регуляризации $\lambda = 0$).

7. МАШИНА ОПОРНЫХ ВЕКТОРОВ

7.1. Принцип машины опорных векторов

Машина опорных векторов – один из мощных методов классификации, по своему дизайну очень похожий на логистическую регрессию, но имеющий от нее существенные отличия, выражающиеся в другом виде гипотезы, функции стоимости, наличии довольно оптимизированного алгоритма минимизации функции стоимости и других.

Название «машина опорных векторов» происходит от перевода английского названия «support vector machine» (SVM). Под таким сокращением метод чаще всего встречается не только в иностранной литературе, но и на русском языке. Далее будем использовать название SVM.

В отличие от логистической регрессии (см. главу 3), функция стоимости для SVM вычисляется как:

$$J(\theta) = C \sum_{i=1}^m [y^{(i)} Cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T x^{(i)})] + \frac{1}{2} \sum_{j=1}^n \theta_j^2.$$

В данном выражении параметр C определяет степень регуляризации модели аналогично логистической регрессии, однако в отличие от параметра логистической регрессии λ , располагается перед первым слагаемым, а потому является обратным к λ (т.е. $C = 1/\lambda$).

Оба слагаемых в функции стоимости не делятся на величину m (размер обучающей выборки). Поскольку при минимизации функции, умножение ее на константу не изменяет результата минимизации, то коэффициент $1/m$ убран из выражения.

Самое главное отличие функции стоимости SVM от логистической регрессии заключается в способе вычисления величины ошибки на одном элементе. В SVM используется две функции: $Cost_1(z)$ и $Cost_0(z)$. Графики этих функции приведены на рис. 7.1.

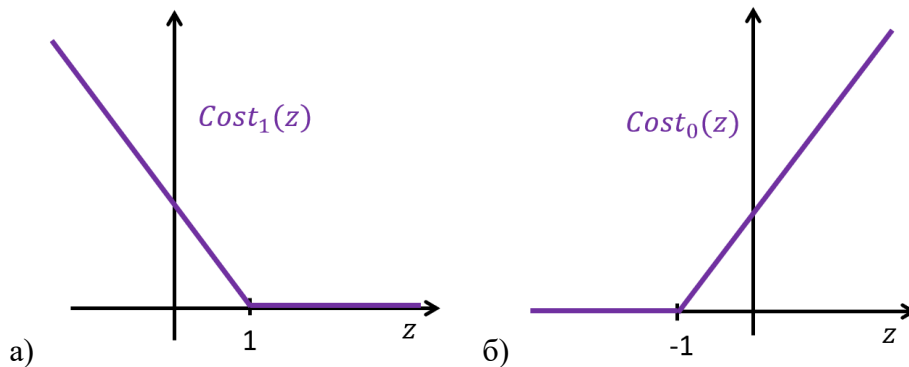


Рис. 7.1

Обе функции являются кусочно-линейными, состоят из двух отрезков прямых, что позволяет быстро вычислять их значение.

Выражение для вычисления ошибки на одном элементе внутри функции стоимости имеет вид:

$$y^{(i)} Cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T x^{(i)}).$$

Аналогично тому, как это было в логистической регрессии, если объект принадлежит классу $y^{(i)} = 1$, то второе слагаемое равно нулю, и ошибка вычисляется по функции $Cost_1(\theta^T x^{(i)})$, а если объект принадлежит классу $y^{(i)} = 0$, то первое слагаемое равно нулю, и ошибка вычисляется по функции $Cost_0(\theta^T x^{(i)})$.

В функции $Cost_1$ и $Cost_0$ в качестве параметра передается:

$$\theta^T x^{(i)} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \dots \\ \theta_n \end{bmatrix}^T \times \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \dots \\ x_n^{(i)} \end{bmatrix} = \theta_0 + \theta_1 x_1^{(i)} + \dots + \theta_n x_n^{(i)}.$$

Функция сигмоиды в SVM не используется.

Как видно по рис. 7.1, величина ошибки будет нулевой, если для элемента класса $y^{(i)} = 1$ величина $\theta^T x^{(i)} \geq 1$ (см. рис. 7.1, а), а для элемента класса $y^{(i)} = 0$ величина $\theta^T x^{(i)} \leq -1$ (см. рис. 7.1, б).

Таким образом задача обучения SVM, т.е. нахождения оптимального значения вектора параметров θ^* , при которого достигается минимум функции стоимости $J(\theta)$, является задачей оптимизации:

$$\theta^* = \min_{\theta} J(\theta).$$

Для решения этой задачи есть хорошие методы, оптимизированные специально для SVM, во многих математических библиотеках на различных языках программирования. Суть этих методов достаточно сложна и не будет рассматриваться в данном курсе. Для обучения модели SVM рекомендуется воспользоваться этими готовыми библиотечными решениями.

После обучения модели и получения оптимального вектора параметров θ , предсказание класса для любого произвольного объекта x выполняется следующей простой гипотезой:

$$h_{\theta}(x) = \begin{cases} 1, & \text{если } \theta^T x \geq 0, \\ 0, & \theta^T x < 0. \end{cases}$$

То есть SVM выдает точный ответ, в отличие от логистической регрессии.

7.2. Классификатор с большими отступами

Рассмотрим задачу классификации с *линейно разделимыми классами*. Таким являются задачи, в которых граница решение в виде линейной функции четко и без исключений отделяет два класса друг от друга. Например, на рис. 7.2 изображен пример линейно разделимых классов для двумерного случая. Прямыми линиями показаны различные варианты границы решения для этих классов.

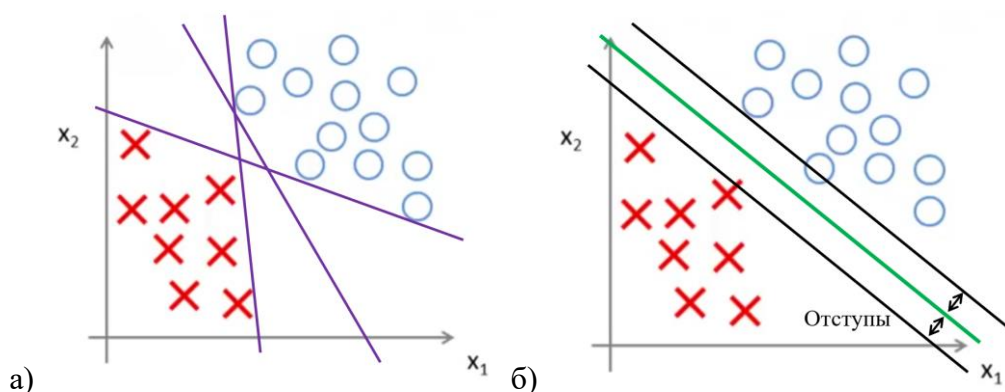


Рис. 7.2

Рассмотрим один частный случай – модель SVM без регуляризации с параметром C заданным большим числом (например, $C \approx 100000$ и более). Это аналогично заданию очень малого λ для логистической регрессии, т.е. фактическому выключению слагаемого регуляризации. Для SVM мы не можем просто убрать это слагаемое, поскольку оно по-прежнему играет значимую роль, что будет видно позднее.

Задача обучения SVM формулируется, как было сказано выше, следующим образом:

$$\min_{\theta} C \sum_{i=1}^m [y^{(i)} Cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) Cost_0(\theta^T x^{(i)})] + \frac{1}{2} \sum_{j=1}^n \theta_j^2.$$

Если значение C довольно большое, это заставляет процедуру оптимизации подбирать такие значения вектора θ , чтобы свести первое слагаемое к наименьшему значению. Функции $Cost_1$ и $Cost_0$ созданы таким образом (см. рис. 7.1), что можно свести значение первого слагаемого к нулю при достаточно большом длине вектора θ . Таким образом, задача превратится в следующую задачу условной оптимизации:

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2,$$

при выполнении условий:

$$\begin{aligned} \theta^T x^{(i)} &\geq 1, \text{ если } y^{(i)} = 1, \\ \theta^T x^{(i)} &\leq -1, \text{ если } y^{(i)} = 0, \text{ для всех } i = 1, \dots, m. \end{aligned}$$

В результате решения этой задачи будет получен вектор θ , которому соответствует граница решения, показанная на рис. 7.2, б. Ее особенностью является расположение таким образом, чтобы расстояние от нее до ближайших объектов обоих классов было максимальным. Это создает дополнительный запас надежности распознавания. Расстояния от границы решения до объектов выборки называют *отступами* (margin), а сам алгоритм SVM иногда называют классификатором с большими отступами. Почему алгоритм формирует именно такую границу, объясняется в следующем разделе.

7.3. Объяснение работы машины опорных векторов

Рассмотрим операцию перемножения двух векторов:

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}.$$

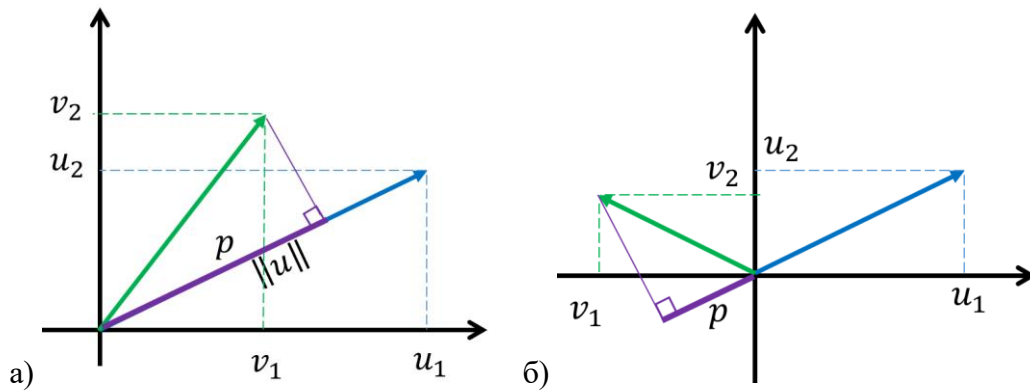
Согласно правилам линейной алгебры, их произведение будет числом:

$$u^T v = [u_1 \quad u_2] \times \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = u_1 v_1 + u_2 v_2 \in \mathbb{R}.$$

Рассмотрим геометрическую интерпретацию этого произведения. Норма (длина вектора) равна: $\|u\| = \sqrt{u_1^2 + u_2^2}$. Если обозначить p – проекцию вектора v на вектор u , то это же произведение можно найти как:

$$u^T v = p \cdot \|u\|.$$

Это иллюстрирует рис. 7.3, а. Стоит учесть, что проекция p является числом со знаком. В ситуации рис. 7.3, а вектора u и v направлены в одну сторону (угол между ними менее 90 градусов), и величина p положительна. Если угол между векторами будет более 90 градусов, то направление проекции p противоположно вектору u , и значение p будет отрицательным (см. рис. 7.3, б), а значит отрицательным и произведение $u^T v$.



а)
Рис. 7.3

б)

Продолжим рассмотрение примера с большим значением C из предыдущего раздела:

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2,$$

при выполнении условий:

$$\begin{aligned} \theta^T x^{(i)} &\geq 1, \text{ если } y^{(i)} = 1, \\ \theta^T x^{(i)} &\leq -1, \text{ если } y^{(i)} = 0, \text{ для всех } i = 1, \dots, m. \end{aligned}$$

Разберем, как влияют данные условия на решение.

Пусть задача у нас двумерная, тогда вектора θ и x имеют вид:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}, \quad x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix}.$$

Для наглядности отбросим фиктивную единицу и будем использовать вектора длиной 2 (это приводит всего лишь к тому, что в нашем примере все вектора будут выходить из начала координат):

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}.$$

Выражение, которое подвергается минимизации, фактически является длиной вектора θ в квадрате:

$$\frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} (\theta_1^2 + \theta_2^2) = \frac{1}{2} \left(\sqrt{\theta_1^2 + \theta_2^2} \right)^2 = \frac{1}{2} \|\theta\|^2.$$

Как было рассмотрено выше, произведение векторов θ и x можно представить, как произведение длины вектора θ на проекцию вектора x на вектор θ :

$$\theta^T x = p \cdot \|\theta\|.$$

В новом представлении задача приобретает следующий вид:

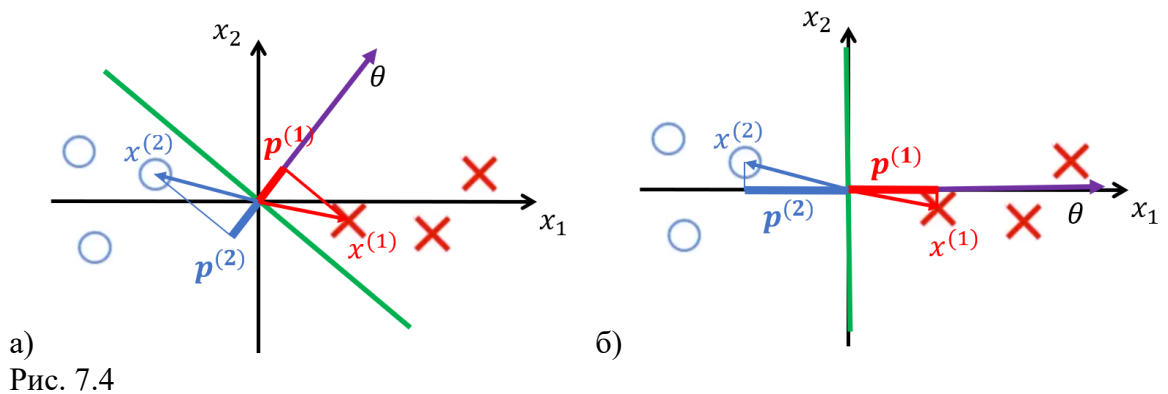
$$\min_{\theta} \frac{1}{2} \|\theta\|^2,$$

при выполнении условий:

$$\begin{aligned} p^{(i)} \|\theta\| &\geq 1, \text{ если } y^{(i)} = 1, \\ p^{(i)} \|\theta\| &\leq -1, \text{ если } y^{(i)} = 0, \text{ для всех } i = 1, \dots, m, \end{aligned}$$

где $p^{(i)}$ – проекция (со знаком) вектора $x^{(i)}$ на вектор θ .

Пусть имеется обучающая выборка с $m = 6$, три точки класса 1 и три точки класса 0 (см. рис. 7.4). Для решения задачи нужно вектор θ наименьшей длины, но при этом должны выполняться условия.



Пусть вектор θ направлен так, как показано на рис. 7.4, а. Граница решения будет располагаться перпендикулярно ему. На минимальную длину вектора θ влияют ограничения: она должна быть такой, чтобы выполнялись условия.

Пусть первая точка $x^{(1)}$ принадлежит классу 1. Ее проекция $p^{(1)}$ на вектор θ не будет большим числом, так как угол между векторами $x^{(1)}$ и θ достаточно большой. В таком случае для того, чтобы выполнялось условие $p^{(1)}\|\theta\| \geq 1$, длина $\|\theta\|$ должна быть достаточно большой.

Аналогично, если выбрать точку $x^{(2)}$ из класса 0, то ее проекция $p^{(2)}$ тоже не велика (по абсолютному значению, т.к. проекция со знаком минус), и для выполнения условия $p^{(2)}\|\theta\| \geq -1$ также потребуется большое значение $\|\theta\|$.

Другая ситуация показана на рис. 7.4, б. Вектор θ направлен по-другому и проекции $p^{(1)}$ и $p^{(2)}$ являются по абсолютному значению гораздо большими числами. Следовательно, величина $\|\theta\|$ для выполнения условий может быть гораздо меньше.

Из двух ситуаций (рис. 7.4, а и 7.4, б) во втором случае длина вектора $\|\theta\|$ будет гораздо меньше, и это решение является более предпочтительным (ведь задача минимизации – найти вектор θ минимальной длины).

Во втором случае отступы от границы решения получаются гораздо больше, чем в первом (отступы, по сути, равны величине проекций $p^{(1)}$ и $p^{(2)}$ ближайших точек на θ).

Таким образом, в процессе подбора значений вектора θ метод SVM находит решение с максимально возможными отступами.

7.4. Ядра

Выше было рассмотрена в общих чертах работа SVM для линейно разделимых классов. Далее рассмотрим задачу, когда классы линейно неразделимы, как показано на рис. 7.5. Для классификации с такой сложной границей решения метод SVM предполагает использование ядер.

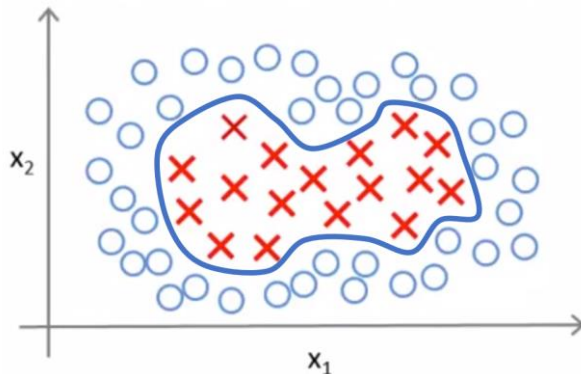


Рис. 7.5

Для этого в функциях $Cost_1$ и $Cost_0$ в качестве параметра вместо полинома $\theta^T x$ используется $\theta^T f$, где f – некий вектор признаков (от англ. слова *features*), вычисляемого для объекта x . Эти признаки вычисляются как значений функций близости, называемый *ядрами*, к некоторым *ключевым точкам* l .

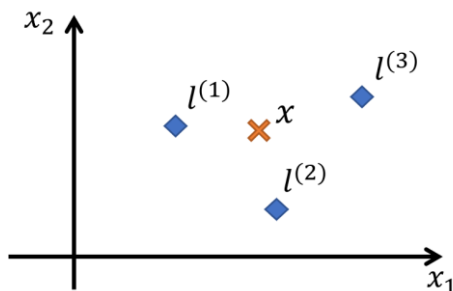


Рис. 7.6

Ключевые точки могут быть заданы произвольно, сгенерированы алгоритмом или в качестве ключевых могут быть выбраны некоторые (или все) точки обучающей выборки (что чаще всего в SVM и применяется).

Ядро или *функция близости* – это специальная функция, которая показывает, насколько две точки близки друг к другу. Один из самых распространенных видов ядра – *ядро Гаусса*:

$$K(x, l^{(j)}) = \exp\left(-\frac{\|x - l^{(j)}\|^2}{2\sigma^2}\right).$$

Функция \exp является возведением числа e в степень аргумента: $\exp(\beta) = e^\beta$.

В верхней части выражения записана норма $\|x - l^{(j)}\|$, что в геометрическом смысле означает расстояние от точки x до точки $l^{(j)}$. Влияние параметра σ рассмотрим позднее.

Например, если объект описывается двумя переменными:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

и заданы три ключевые точки, как показано на рис. 7.6, то у каждого объекта x будет три признака:

$$\begin{aligned} f_1 &= K(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right), \\ f_2 &= K(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right), \\ f_3 &= K(x, l^{(3)}) = \exp\left(-\frac{\|x - l^{(3)}\|^2}{2\sigma^2}\right). \end{aligned}$$

Таким образом, вместо линейной функции $\theta_0 + \theta_1 x_1 + \theta_2 x_2$ в SVM будет использоваться $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3$. Добавляя новые ключевые точки, можно сконструировать функцию большой сложности.

Если объект x близок к ключевой точке $l^{(j)}$ (рис. 7.7, а), тогда расстояние $\|x - l^{(j)}\| \rightarrow 0$ (очень мало), следовательно значение:

$$f_j = \exp\left(-\frac{\sim 0}{2\sigma^2}\right) \approx 1.$$

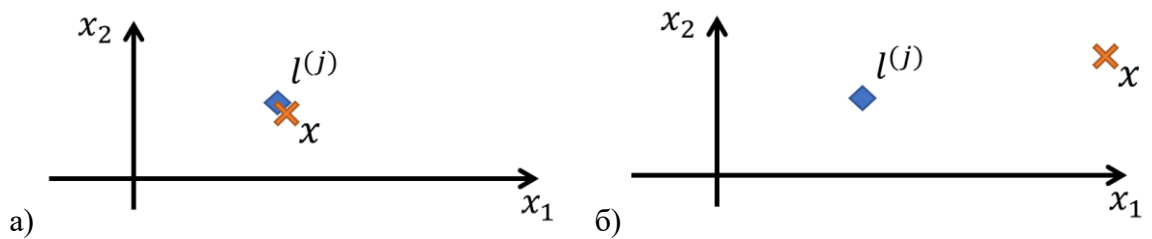


Рис. 7.7

Если объект x расположен далеко от точки $l^{(j)}$ (рис. 7.7, б), тогда расстояние $\|x - l^{(j)}\| \gg 0$, следовательно значение $f_j \ll 1$. Если точка достаточно далеко, то $f_j \approx 0$. В этом смысл ядра как функции близости: чем ближе x к ключевой точке $l^{(j)}$, тем больше значение признака f_j .

Рассмотрим, как влияет расположение ключевых точек на результат классификации. Пусть имеется две ключевые точки класса 1, как показано на рис. 7.8. В таком случае SVM будет работать по следующей гипотезе:

$$h_\theta(x) = \begin{cases} 1, & \text{если } \varphi(x, \theta) \geq 0, \\ 0, & \varphi(x, \theta) < 0, \end{cases}$$

где $\varphi(x, \theta) = \theta_0 + \theta_1 f_1 + \theta_2 f_2$.

Предположим, что SVM была обучена и получен оптимальный вектор параметров:

$$\theta = \begin{bmatrix} -0.5 \\ 1 \\ 1 \end{bmatrix}.$$

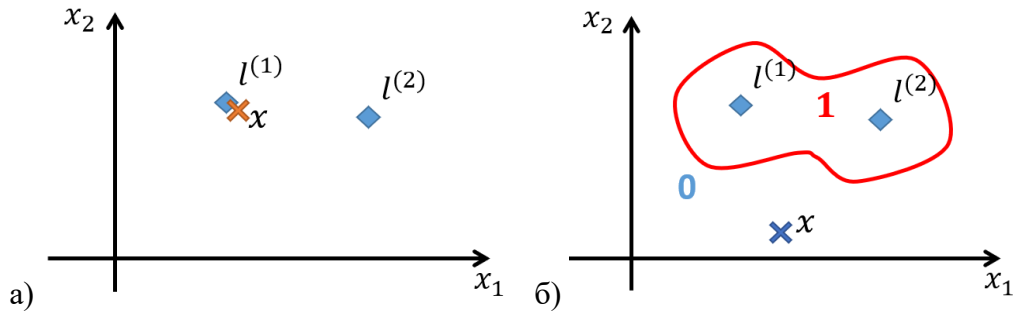


Рис. 7.8

Возьмем точку x близко к ключевой точке $l^{(1)}$ (см. рис. 7.8, а). В этом случае: $f_1 \approx 1$ (точка расположена близко), $f_2 \approx 0$ (вторая точка расположена далеко), тогда:

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 = -0.5 + 1 \cdot 1 + 1 \cdot 0 = 0.5 \geq 0.$$

Следовательно, точка x будет отнесена к классу 1.

Возьмем точку x далеко от обеих ключевых точек (см. рис. 7.8, б). В этом случае: $f_1 \approx 0$ и $f_2 \approx 0$, тогда:

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 = -0.5 + 1 \cdot 0 + 1 \cdot 0 = -0.5 < 0.$$

Следовательно, в этом случае точка x будет отнесена к классу 0.

В целом, в окрестности ключевых точек можно выделить зоны, близкое к ним, в которых любой объект будет относиться к тому классу, к которому и ключевая точка. На рис. 7.8, б эта зона обведена красным. Внутри этой зоны объекты будут относиться к классу 1, а снаружи – к классу 0.

В самой распространенной реализации SVM с ядром в качестве ключевых точек выбираются все точки обучающей выборки:

$$l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}.$$

Для произвольного x :

$$\begin{aligned} f_1 &= K(x, l^{(1)}), \\ f_2 &= K(x, l^{(2)}), \\ &\dots \\ f_m &= K(x, l^{(m)}). \end{aligned}$$

В результате получаем вектор признаков размером $m + 1$:

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \dots \\ f_m \end{bmatrix} \in \mathbb{R}^{m+1},$$

где $f_0 = 1$ – фиктивный элемент.

Гипотеза SVM с ядром имеет вид:

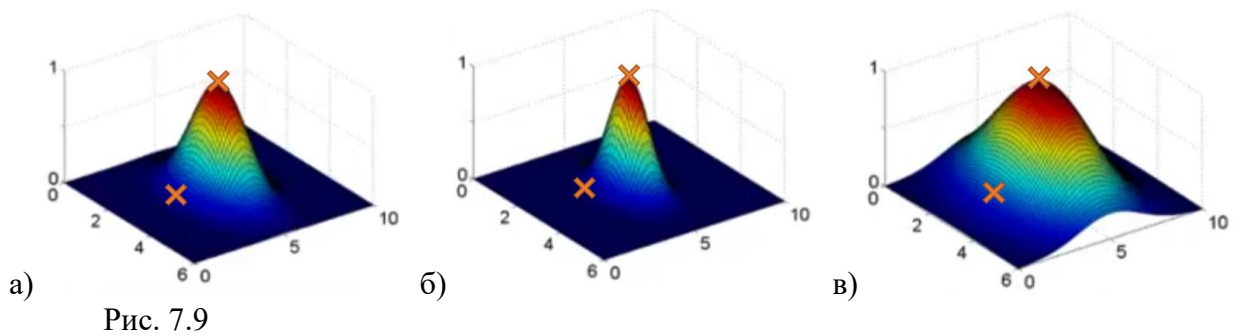
$$h_\theta(x) = \begin{cases} 1, & \text{если } \theta^T f \geq 0, \\ 0, & \theta^T f < 0. \end{cases}$$

Функция стоимости для SVM с ядром:

$$J(\theta) = c \sum_{i=1}^m [y^{(i)} \text{Cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{Cost}_0(\theta^T f^{(i)})] + \frac{1}{2} \sum_{j=1}^m \theta_j^2.$$

Эта функция имеет отличия от функции стоимости SVM без ядра: в функции Cost передается результат $\theta^T f^{(i)}$ (признаков, вычисляемых ядром), размер вектора признаков f

и вектора параметров θ равен t (объему обучающей выборки).



Рассмотрим влияние параметра σ на вычисление признаков объектов. На рис. 7.9, а показан график значения ядра Гаусса при $\sigma = 1$. Если точка x равна ключевой точке, то значение ядра будет равно 1 (верхняя точка на рисунке). Если точка отстоит от ключевой на некоторое расстояние (точка ниже и левее на рисунке), то значение ядра будет меньше 1.

Если задать меньшее значение σ , например, $\sigma = 0.5$, значения ядра будут меньшими, график становится уже (рис. 7.9, б). Если точка x равна ключевой точке, то значение ядра по-прежнему равно 1, но для второй точки значение будет меньшим, чем при ядре с $\sigma = 1$.

Если задать большее значение σ , например, $\sigma = 3$, значения ядра будут большими, график становится шире (рис. 7.9, в). Если точка x равна ключевой точке, то значение ядра по-прежнему равно 1. Для второй точки значение самым большим из всех рассмотренных случаев на рис. 7.9.

Таким образом, параметром σ можно задавать размер окрестности ключевой точки, на которую она оказывает существенное влияние.

7.5. Использование машины опорных векторов

Метод SVM во многом похож на метод логистической регрессии. За счет оптимизированных процедур обучения, использования ядер и других особенностей SVM, во многих задачах он превосходит логистическую регрессию по точности и быстродействию.

Пусть n – количество признаков объекта x ($x \in \mathbb{R}^n$ или $x \in \mathbb{R}^{n+1}$, если добавлен фиктивный элемент); t – объем обучающей выборки.

Если n намного больше, чем t , то рекомендуется использовать SVM без ядра (или логистическую регрессию).

Если n меньше t , но объем выборки не очень большой (порядка нескольких десятков тысяч элементов), то лучше всего подойдет SVM с ядром.

В случае, если n меньше t , и объем выборки очень велик (от 50 тысяч и более), то SVM с ядром может работать достаточно медленно (вектор признаков будет очень большой). В этой ситуации рекомендуется конструировать новые переменные и использовать SVM без ядра (или логистическую регрессию).

Если используется SVM с ядром, то чаще всего применяется ядро Гаусса. Однако для SVM существуют и другие виды ядер: полиномиальное, Хи-квадрат (χ^2) и другие. Есть также специфические ядра для определенных типов входных данных, например, ядро строкового расстояния для обработки текстов. Также можно встретить такое понятие, как «линейное ядро», что в действительности означает, что ядро для SVM не используется, и вычисляется функция $\theta^T x$.

Перед обучением SVM необходимо выбрать значение для параметра C , а в случае, если используется ядро, то и параметры ядра (σ для ядра Гаусса).

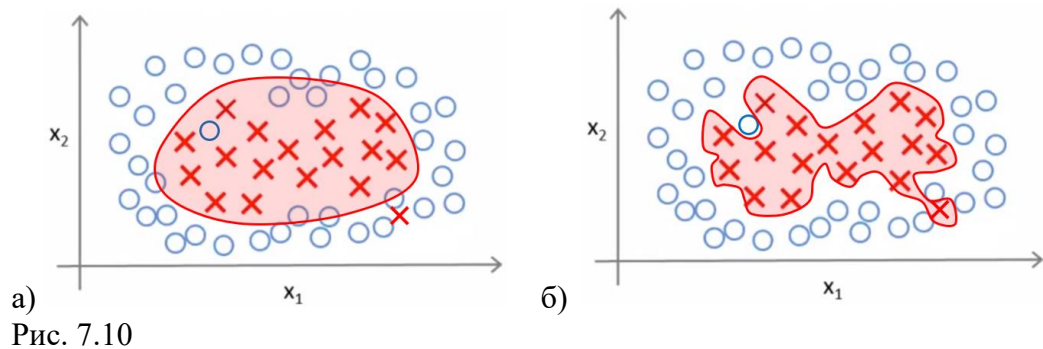
Как было описано в главе 6, подбор параметров рекомендуется делать с использованием валидационной выборки, а не на обучающем наборе.

Смысл параметра C аналогичен параметру λ регуляризованной логистической регрессии, но влияние его обратное.

Если задать большие значения C , то модель будет более склонна к дисперсии и менее склонна к смещению, повышается риск формирования переобученной модели. При малых значениях C влияние слагаемого регуляризации возрастает, модель менее склонна к дисперсии и больше склонна к смещению, возрастает риск недообученной модели.

При использовании ядра Гаусса с большими значениями параметра σ также модель менее склонна к дисперсии и больше склонна к смещению. Это возникает вследствие того, что широкие окрестности ключевых точек накладываются друг на друга и размываются (см. рис. 7.10, а).

При малых значениях σ модель будет более склонна к дисперсии и менее склонна к смещению, повышается риск формирования переобученной модели (см. рис. 7.10, б). Также модель становится более чувствительна к выбросам (ошибкам в обучающей выборке).

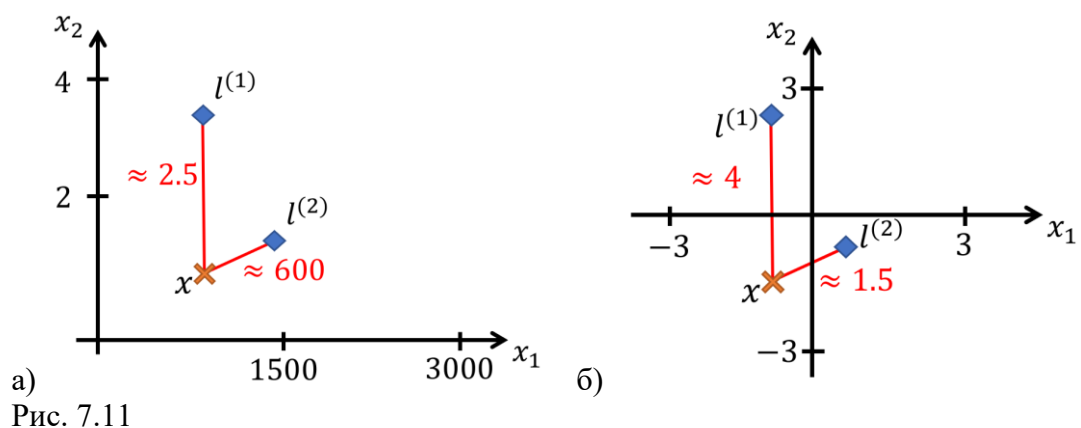


Еще одним важным фактором при использовании SVM с ядром является нормализация исходных данных. В некоторых задачах масштаб значений переменных, описывающих объекты, может различаться на порядки. Например, при описании объекта недвижимости, переменная площади дома может принимать значения до сотен и тысяч квадратных метров, а переменная количества комнат или этажей – единицы и десятки.

Рассмотрим пример на рис. 7.11. Переменные могут принимать значения в следующих диапазонах: $x_1 \in [0, 3000]$, $x_2 \in [1, 5]$. При вычислении признаков модели ядром Гаусса по формуле:

$$K(x, l^{(j)}) = \exp\left(-\frac{\|x - l^{(j)}\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\sum_{i=1}^n (x_i - l_i^{(j)})^2}{2\sigma^2}\right)$$

мы обнаружим, что близость к ключевой точке очень слабо зависит от значения x_2 и почти целиком определяется по x_1 (см. рис. 7.11, а).



Это приводит к ошибкам классификации. Для устранения этой проблемы необходимо выполнить нормализацию данных, которая обсуждалась в разделе 2.2.

Еще один распространенный вариант нормализации в современном машинном обучении определяется следующей формулой:

$$\hat{x}_i = \frac{x_i - \mu_i}{s_i},$$

где μ_i – среднее значение (математическое ожидание) переменной x_i ; s_i – стандартное отклонение переменной x_i .

В результате диапазон значений переменных будет сопоставим. На рис. 7.11, б показан вариант с нормализованными данными.

Рассмотренный метод SVM позволяет выполнить классификацию на два класса. Для решения многоклассовой задачи применяется тот же подход «один-против-всех», который разбирался для логистической регрессии в разделе 3.6.

Для решения задачи классификации на M классов ($y \in \{1, 2, \dots, M\}$) строится M классификаторов, каждый из которых выделяет свой класс. В результате обучения будет получено M векторов параметров: $\theta^{(1)}, \theta^{(2)}, \dots, \theta^{(M)}$.

Результат классификации определяется по максимальному значению линейной функции $(\theta^{(k)})^T x$:

$$h(x) = \underset{k}{\operatorname{argmax}} (\theta^{(k)})^T x.$$

Также стоит отметить, что «один-против-всех» для SVM приводит к формированию «классификатора с отказами», то есть возможна ситуация, что все M функций выдали отрицательное значение. В таком случае результат классификации не определен.

Вопросы и задания

1) Рассмотрим задачу минимизации следующих двух выражений:

$$\min_{\theta} \frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2$$

$$\min_{\theta} C \sum_{i=1}^m [y^{(i)} \operatorname{Cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \operatorname{Cost}_0(\theta^T x^{(i)})] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

В каком случае решение этих двух задач даст в ответ одинаковое значение θ ?

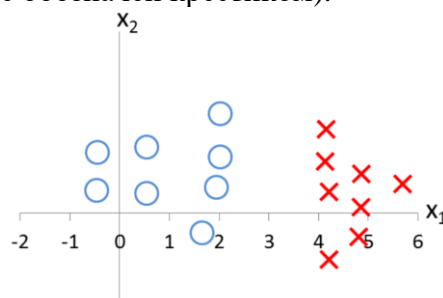
а) $C = \lambda$

б) $C = -\lambda$

в) $C = \frac{1}{\lambda}$

г) $C = \frac{2}{\lambda}$

2) Рассмотрим задачу классификации на два класса: $y = 0$ (на рисунке обозначен кружком) и $y = 1$ (на рисунке обозначен крестиком).



Вы обучаете SVM с параметром $C = 100\,000$ и условием:

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0.$$

Какие значения могут быть получены для θ_0 , θ_1 и θ_2 ?

- а) $\theta_0 = 3, \theta_1 = 1, \theta_2 = 0$
- б) $\theta_0 = -3, \theta_1 = 1, \theta_2 = 0$
- в) $\theta_0 = 3, \theta_1 = 0, \theta_2 = 1$
- г) $\theta_0 = -3, \theta_1 = 0, \theta_2 = 1$

3) Рассмотрим задачу классификации на два класса: $y = 0$ (на рисунке обозначен кружком) и $y = 1$ (на рисунке обозначен крестиком).



Задача обучения SVM задана как:

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

при условии:

$$\begin{aligned} p^{(i)} \|\theta\| &\geq 1, \text{ если } y^{(i)} = 1, \\ p^{(i)} \|\theta\| &\leq -1, \text{ если } y^{(i)} = 0, \end{aligned}$$

где $p^{(i)}$ – проекция $x^{(i)}$ на θ .

В оптимальном значении θ какой будет величина $\|\theta\|$?

- а) $\|\theta\| = 1/4$
- б) $\|\theta\| = 1/2$
- в) $\|\theta\| = 1$
- г) $\|\theta\| = 2$

4) Вы обучаете SVM с ядром Гаусса и обнаружили, что модель переобучается. Что стоило бы попробовать для устранения этой проблемы?

- а) увеличить C
- б) уменьшить C
- в) увеличить σ
- г) уменьшить σ

5) Вы обучаете SVM с ядром Гаусса и хотите подобрать значения для параметров C и σ . Как будет правильно это сделать?

- а) Выбрать значения C и σ , на которых достигается лучший результат на обучающей выборке.
- б) Выбрать значения C и σ , на которых достигается лучший результат на валидационной выборке.
- в) Выбрать значения C и σ , на которых достигается лучший результат на тестовой выборке.
- г) Выбрать значения C и σ , на которых достигается наибольшая величина отступа SVM.