

Лекция 3. Градиентный спуск для обучения
дифференцируемых моделей машинного обучения.
Методы регуляризации моделей машинного обучения.
Линейная регрессия.

Глинский А.В.

Цель занятия: В этом модуле мы познакомимся с такими темами, как градиентный спуск для обучения дифференцируемых моделей машинного обучения, регуляризация моделей машинного обучения для борьбы с переобучением, а также с алгоритмом линейной регрессии. После прохождения модуля ученики смогут понимать основные принципы использования градиентного спуска и его модификаций для решения задач оптимизации функции потерь в задачах машинного обучения, применять регуляризацию для предотвращения переобучения и использовать алгоритм линейной регрессии для решения задач регрессии.

Содержание

0.1	Градиентный спуск	3
0.1.1	Функции потерь и функционалы ошибки в регрессии	3
0.1.2	Градиент функции потерь	4
0.1.3	Определение градиентного спуска	7
0.1.4	Параметры градиентного спуска	8
0.1.5	Модификации градиентного спуска	10
0.1.6	Вопросы для самопроверки	16
0.1.7	Резюме по разделу	17
0.2	Регуляризация	19
0.2.1	Основные методы регуляризации	19
0.2.2	Методы регуляризации, основанные на добавлении слагаемого к функции потерь модели	19
0.2.3	Вопросы для самопроверки	23
0.2.4	Резюме по разделу	23
0.3	Линейная регрессия	24
0.3.1	Визуальная демонстрация алгоритма	24
0.3.2	Описание алгоритма	26
0.3.3	Подготовка данных для алгоритма	27
0.3.4	Процесс обучения	27
0.3.5	Оценка качества алгоритма	30
0.3.6	Интерпретация признаков с помощью алгоритма	30
0.3.7	Модификации алгоритма	31
0.3.8	Область применения алгоритма	31
0.3.9	Плюсы и минусы алгоритма	32
0.3.10	Реализация алгоритма в Python	32
0.3.11	Вопросы для самопроверки	33
0.3.12	Резюме по разделу	33
0.4	Резюме по модулю	34

0.1 Градиентный спуск

Мы уже познакомились с таким методом машинного обучения, как метод ближайших соседей. Это очень простой метод, обучение в котором, по сути, сводится к запоминанию обучающей выборки. На этом занятии мы приступим к изучению дифференцируемых моделей линейной регрессии, которые обучаются с помощью градиентного спуска. Давайте разберемся, что это такое.

Градиентный спуск - это метод, который помогает модели машинного обучения находить оптимальные значения параметров. Он делает это путем постепенного изменения параметров таким образом, чтобы функция потерь (разница между предсказанными значениями и реальными значениями целевой переменной), становилась все меньше и меньше. Градиентный спуск позволяет найти оптимальные значения параметров модели, минимизируя функцию потерь. Благодаря этому методу можно обучать различные модели машинного обучения, такие как линейные модели, нейронные сети и другие.

Теперь рассмотрим этот процесс подробнее.

0.1.1 Функции потерь и функционалы ошибки в регрессии

И начнем мы наше знакомство с градиентным спуском с таких понятий, как функции потерь и функционалы ошибок в задачах регрессии.

Функция потерь (loss function) в задачах регрессии используется для оценки того, насколько хорошо модель предсказывает целевую переменную. Функция потерь выражает разницу между предсказанными значениями и реальными значениями целевой переменной на конкретном обучающем примере.

В задачах регрессии наиболее популярные функции потерь — это квадратичная ошибка, абсолютная ошибка и функция потерь Хьюбера (Рисунок 1).



Рисунок 1: Популярные функции потерь в задачах регрессии

Функции потерь используются при обучении моделей для определения оптимальных значений параметров модели, которые минимизируют потери. Для определения минимума функции удобно использовать производную функции (Рисунок 2).

На рисунке 2 мы стартуем из произвольной точки, вычисляем в ней производную и движемся в сторону, обратную значению производной, со скоростью, указанной в параметре «learning rate». Траектория движения указана красными линиями. Например, для направленной вверх параболы (квадратичная функция), которая является гладкой функцией с одним

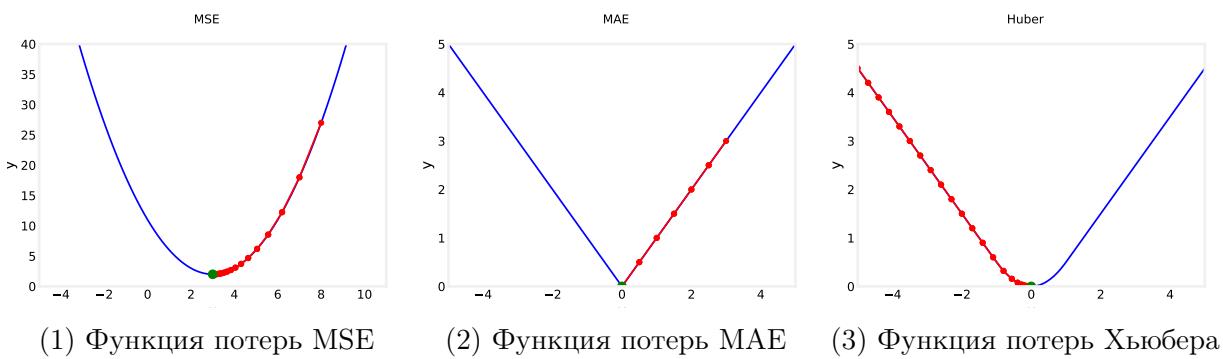


Рисунок 2: Нахождение минимума функций с помощью производной

глобальным минимумом, производная указывает направление роста функции, соответственно, двигаясь в обратном направлении, можно достичь глобального минимума. Это справедливо и для абсолютной функции потерь. Однако стоит заметить, что в абсолютной функции потерь производная не определена в нуле. Одним из способов решения этой проблемы является использование функции потерь Хьюбера.

Давайте введем еще одно понятие — функционал ошибки. **Функционал ошибки (cost function)** — это математическая функция, которая измеряет среднюю ошибку модели на всем наборе данных. Функционал ошибки представляет собой сумму функции потерь на всех примерах обучающих данных, нормализованную на количество примеров в наборе данных. Функционал ошибки показывает, как хорошо модель работает на всех данных в целом и служит основной метрикой для оценки качества модели.

Основное отличие между функцией потерь и функционалом ошибки заключается в том, что функция потерь измеряет ошибку на каждом примере данных, в то время как функционал ошибки измеряет среднюю ошибку на всем наборе данных. В задачах регрессии наиболее популярные функционалы потерь — это среднеквадратичная ошибка (MSE), средняя абсолютная ошибка (MAE) и функционал ошибки Хьюбера.

0.1.2 Градиент функции потерь

Для минимизации функции потерь, включающей в себя более чем один параметр, в машинном обучении используется градиент — вектор частных производных, который указывает направление наискорейшего возрастания функции потерь относительно параметров модели. Он используется для обновления параметров модели во время обучения с помощью метода градиентного спуска. Градиент функции потерь $L(\mathbf{x} \cdot \mathbf{w}, \hat{\mathbf{y}})$ обозначается с помощью символа «набла»: $\nabla L(\mathbf{x} \cdot \mathbf{w}, \hat{\mathbf{y}})$.

Градиент в машинном обучении может быть вычислен с помощью алгоритмов дифференцирования, таких как автоматическое дифференцирование или символьное дифференцирование. Важно отметить, что в некоторых случаях функция потерь может быть не дифференцируемой или иметь разрывы, что усложняет вычисление градиента и требует использования альтернативных методов оптимизации. В регрессии, например, абсолютная функция потерь недифференцируема в нуле. Для решения этой проблемы используют функцию потерь Хьюбера, где для малых значений функция потерь вычисляется через квадратичную ошибку, а для значений, больших δ — через абсолютную ошибку. В функции потерь Хьюбера существует гиперпараметр δ , отвечающий за переход от подсчета квадратичной ошибки к абсолютной.

Часто в литературе встречается значение $\delta = 1$. Но это значение не является универсальным, и его следует выбирать, исходя из характеристик данных и требуемой чувствительности к выбросам. Градиент – это вектор, показывающий направление наискорейшего возрастания функции в заданной точке. В машинном обучении градиент используется для настройки параметров модели с помощью методов оптимизации, таких как градиентный спуск или его модификации.

Некоторые из свойств градиента в машинном обучении:

- С помощью градиента можно находить точки экстремума функции (локальные минимумы и максимумы). Экстремум функции – это точка на ее графике, в которой функция достигает наибольшего (максимум) или наименьшего (минимум) значения в данной области определения (Рисунок 3). Нахождение локальных и глобального минимумов функции потерь является основной целью использования градиента в машинном обучении. Если функция дифференцируема в экстремуме, то градиент в этой точке равен

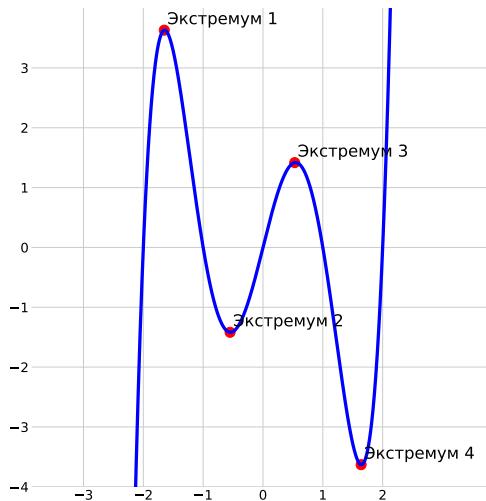


Рисунок 3: Экстремумы функции

нулю. Соответственно, с помощью градиента можно искать экстремумы функции. Можно делать это аналитически. Однако в случае сложных функций (Рисунок 4), какими зачастую являются функции потерь в машинном обучении, используются численные методы нахождения минимальных значений функции.

- Направление наискорейшего убывания: градиент показывает направление наибольшего увеличения функции в заданной точке. Это позволяет использовать градиент для настройки параметров модели с целью уменьшения функционала ошибки. Для этого используется антиградиент – значение градиента с обратным знаком (Рисунок 5). Для объяснения использования градиента для выявления направления наискорейшего убывания обычно пользуются метафорой горы. Представьте себе, что вы стоите на вершине горы и хотите спуститься в долину. Вы хотите найти точку с наименьшей высотой долины и двигаться в этом направлении. Вы можете смотреть в любом направлении и определять, куда надо двигаться, чтобы спуститься вниз. Градиент функции – это инструмент, который позволяет понять, в каком направлении двигаться для достижения минимального значения функции. Если градиент функции большой, это значит, что

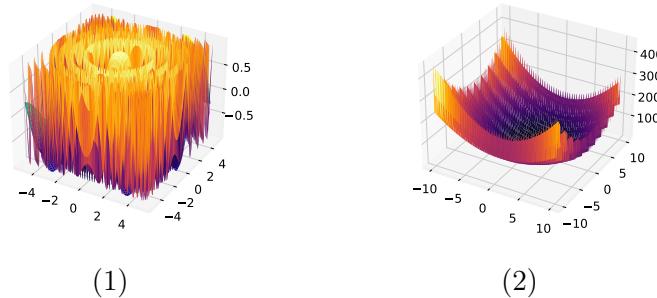


Рисунок 4: Сложные функции потерь

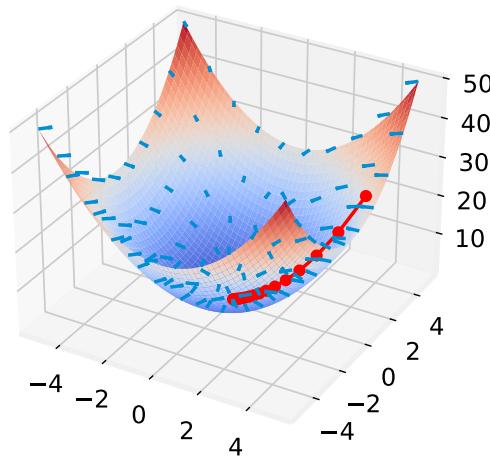


Рисунок 5: Использование антиградиента для нахождения минимума функции

находитесь на крутом склоне горы, и вам нужно двигаться быстрее, чтобы спуститься вниз. Если градиент маленький, это значит, что вы находитесь на пологом участке горы, и двигаться можно медленно (Рисунок 6).

- Норма градиента связана со скоростью обучения: чем больше градиент, тем быстрее модель обучается. Однако, слишком большой градиент может привести к неустойчивости процесса оптимизации и переобучению модели.
- Градиент может быть вычислен аналитически: в некоторых случаях, градиент функции может быть выражен в явном виде, что упрощает процесс оптимизации. Например, для линейной регрессии, градиент можно выразить в явном виде в случае использования среднеквадратичной ошибки в качестве функционала ошибки.
- Градиент может быть вычислен численно: в случаях, когда аналитический градиент не может быть вычислен, можно использовать численное дифференцирование для приближенного вычисления градиента.
- Градиент направлен перпендикулярно линии уровня в определенной точке функции. Линия уровня функции - это множество точек в ее области определения, в которых



Рисунок 6: Аналогия градиента и спуска с горы

значение функции остается постоянным, т.е. все точки на линии имеют одинаковое значение функции (Рисунок 7).

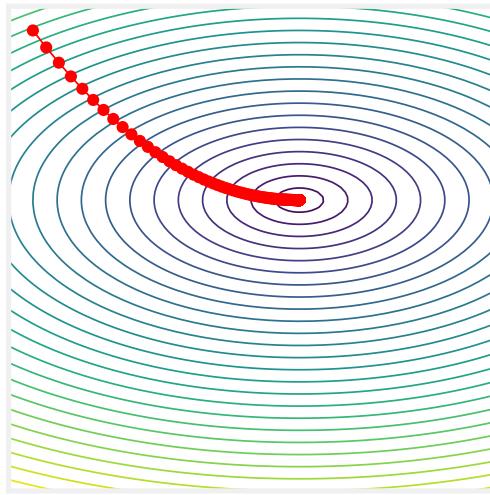


Рисунок 7: Градиент перпендикулярно линии уровня

0.1.3 Определение градиентного спуска

Теперь мы познакомились со всеми понятиями, необходимыми для определения градиентного спуска. Градиентный спуск - это метод оптимизации, используемый в машинном обучении для нахождения локального минимума функции потерь модели. Он основан на вычислении градиента функции потерь по параметрам модели и последующем обновлении этих параметров в направлении наиболее быстрого убывания градиента.

Вот общие шаги алгоритма градиентного спуска:

1. Инициализация: Выберите начальное значение параметров модели (весов) случайным образом или на основе некоторой предварительной информации.
2. Вычисление градиента: Вычислите градиент функции потерь (производной функции по каждому параметру модели). Градиент показывает направление наиболее быстрого возрастания функции.
3. Обновление параметров: Используя градиент, обновите значения параметров модели, перемещаясь в направлении, противоположном градиенту. Это делается путем вычисления некоторого шага обучения (learning rate) умноженного на градиент от текущих значений параметров.
4. Повторение шагов: Повторяйте шаги 2 и 3 до достижения критерия остановки. Обычно условием остановки является достижение максимального числа итераций, достижение требуемой точности или сходимость функции потерь.
5. Возврат результата: Возвращение обновленных значений параметров модели, которые являются локальным минимумом функции потерь.

Это общие шаги, и могут существовать различные вариации и модификации алгоритма градиентного спуска, такие как стохастический градиентный спуск или мини-пакетный градиентный спуск. Градиентный спуск является одним из основных методов оптимизации в машинном обучении, который может использоваться для настройки параметров любых дифференцируемых моделей.

0.1.4 Параметры градиентного спуска

Параметры градиентного спуска включают:

- Скорость обучения (learning rate): это гиперпараметр, который определяет размер шага в каждой итерации обновления параметров. Если скорость обучения выбрана слишком большой, то метод может не сойтись, и оптимизация не будет достигнута. Если скорость обучения слишком мала, метод может занять слишком много времени на поиск оптимума. Выбор оптимальной длины шага (шага градиентного спуска) в градиентном спуске - это важный вопрос, который может существенно влиять на скорость и точность сходимости алгоритма. Вот несколько способов выбора длины шага в градиентном спуске:
 - Адаптивный шаг (adaptive step): длина шага изменяется по формуле или в зависимости от ситуации на каждой итерации. Например, можно использовать методы, такие как AdaGrad, RMSprop или Adam, которые адаптивно изменяют длину шага в зависимости от градиента и предыдущих шагов. Приведем пример базовой формулы для изменения скорости обучения (Формула 1):

$$\alpha_t = \alpha_0 \cdot \frac{1}{1 + d \cdot t},$$

где α_t - скорость обучения в момент времени t ,

α_0 - начальная скорость обучения,

d - коэффициент затухания скорости обучения,

t - текущая эпоха обучения.

(1)

- Фиксированный шаг (constant step): длина шага остается постоянной в течение всего процесса обучения. Этот метод прост в реализации, но может быть неэффективным, так как шаг может быть слишком маленьким или слишком большим, что приведет к медленной сходимости или неустойчивости.
- Backtracking line search: это метод, который на каждой итерации градиентного спуска ищет оптимальную длину шага, которая удовлетворяет определенным критериям. Например, можно использовать метод Армихо (Armijo), который гарантирует уменьшение функции потерь на каждом шаге, или метод золотого сечения (golden section), который находит оптимальную длину шага в заданном интервале.
- Функция потерь (loss function): это функция, которая измеряет разницу между текущими предсказаниями и правильными ответами. Оптимизационный алгоритм использует градиент функции потерь для определения направления, в котором нужно обновлять параметры.
- Метод инициализации параметров: это способ задания начальных значений параметров. Он может влиять на скорость сходимости метода, так как некоторые методы инициализации могут быть более эффективными, чем другие. Некоторые из наиболее распространенных методов инициализации включают в себя:
 - Случайная инициализация - начальные значения параметров модели выбираются случайным образом. Это является наиболее распространенным методом инициализации.
 - Инициализация нулями - начальные значения всех параметров устанавливаются в ноль. Этот метод может быть полезным, если модель уже имеет предварительно обученные веса, которые затем можно дообучить на новых данных.
 - Инициализация на основе распределения - начальные значения параметров выбираются из определенного распределения, такого как нормальное или равномерное. Этот метод может быть полезным для определенных типов моделей и задач.

Если начальные значения слишком далеки от оптимальных, то метод может потребовать больше времени на достижение оптимума. Для решения проблемы застревания в неоптимальных локальных минимумах существует техника «мультистарта». Мультистарт (англ. «multistart») - это метод инициализации весов модели, который заключается в запуске обучения модели несколько раз с разными начальными значениями весов. Использование мультистарта может улучшить качество обучения модели и уменьшить вероятность застревания в локальных оптимумах.

- Количество итераций: это количество шагов обновления параметров, которые нужно выполнить, чтобы достичь оптимума. Это может быть фиксированный параметр или зависеть от критерия сходимости.
- Критерии остановки: критерии остановки градиентного спуска определяют, когда следует остановить итерации алгоритма оптимизации. Это важно для того, чтобы избежать бесконечного цикла и переобучения модели. Вот некоторые распространенные критерии остановки градиентного спуска:
 - Достижение определенного количества итераций. Задается максимальное число итераций, после чего алгоритм останавливается. Количество итераций, необходимых для обучения алгоритма машинного обучения, может значительно различаться в зависимости от многих факторов, таких как размер и сложность набора данных, выбранный алгоритм обучения, используемая аппаратная конфигурация, настройки гиперпараметров и т.д.

В целом, обучение алгоритма машинного обучения требует проведения множества итераций или эпох, где одна эпоха представляет собой один проход по всем обучающим примерам в наборе данных. Часто требуется проводить многократные эпохи для достижения оптимальных результатов обучения.

Например, для некоторых типов нейронных сетей на больших наборах данных, может потребоваться проведение сотен или даже тысяч итераций, прежде чем модель сможет достигнуть высокой точности предсказаний. Однако, для более простых моделей или на более маленьких наборах данных, может потребоваться гораздо меньшее количество итераций (до 100) для достижения приемлемых результатов.

- Достижение определенной точности. Определяется, какой порог разницы между значениями функции потерь на текущей и предыдущей итерациях считать достаточным для остановки алгоритма. Если разница меньше порога, то алгоритм прекращает работу.
- Норма градиента становится меньше заданного значения. Норма градиента определяет скорость изменения функции потерь в каждой точке пространства параметров модели. Если норма градиента становится меньше заданного значения, то это означает, что модель достигла локального минимума, и алгоритм может быть остановлен.
- Сходимость функции потерь. Алгоритм останавливается, когда значения функции потерь перестают существенно меняться на протяжении нескольких итераций.
- Проверка изменения параметров модели. Алгоритм может быть остановлен, когда значения параметров модели перестают изменяться существенно на протяжении нескольких итераций.
- Превышение максимального времени работы. Можно установить максимальное время работы алгоритма, после истечения которого он будет остановлен. Время можно установить в зависимости от сложности задачи и располагаемых вычислительных ресурсов.

Выбор критерия остановки градиентного спуска зависит от конкретной задачи и требуемой точности. Хорошей практикой является комбинирование нескольких критериев, чтобы получить более надежный результат, например, одновременное отслеживание нормы градиента, изменения параметров модели, сходимости функции потерь.

0.1.5 Модификации градиентного спуска

Существуют различные модификации алгоритма градиентного спуска, которые позволяют улучшить сходимость и скорость работы. Рассмотрим некоторые из них:

Модификации с обучением не по всей выборке

Стochastic gradient descent (SGD) и mini-batch gradient descent (Mini-batch GD) являются методами оптимизации для обучения нейронных сетей и других моделей машинного обучения.

SGD используется для обновления весов модели после каждого примера обучающего набора, в то время как Mini-batch GD используется для обновления весов после каждой небольшой группы примеров, называемой мини-батч.

Преимущества модификаций с обучением не по всей выборке

- Они могут быстрее обучать модели, так как обновления происходят на каждом примере/мини-батче.

- Они могут работать с большими обучающими наборами данных, которые не могут поместиться в память компьютера для GD.
- Они могут помочь избежать застревания в локальных оптимумах, поскольку они более случайные, чем обычный градиентный спуск.

Формулы модификаций:

- Стохастический градиентный спуск (SGD). Вместо вычисления градиента по всей выборке данных, SGD вычисляет градиент только по одному обучающему объекту на каждой итерации. Это позволяет ускорить обучение и уменьшить требования к памяти. Формула SGD представлена в уравнении 2.

$$w_{t+1} = w_t - \alpha_t \nabla L(w_t; y_{it}, m(x_{it})),$$

где w_t - вектор параметров модели на шаге t ,

α_t - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

∇L - градиент функции потерь $L(w_t; y_{it}, m(x_{it}))$ по параметрам модели m ,

x_{it} и y_{it} - i -й обучающий пример из обучающего набора данных,

используемый на шаге t .

(2)

- Мини-пакетный градиентный спуск (Mini-batch GD). Это комбинация стохастического градиентного спуска и полного градиентного спуска. На каждой итерации алгоритм вычисляет градиент на подвыборке данных фиксированного размера (например, 32 или 64 элемента), что позволяет улучшить скорость обучения и сходимость. Формула mini-batch GD представлена в уравнении 3.

$$w_{t+1} = w_t - \alpha_t \frac{1}{length(MB)} \sum_{i \in MB} \nabla L(w_t; y_{it}, m(x_{it}))$$

где w_t - вектор параметров модели на шаге t ,

α_t - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

MB - mini-batch (мини-пакет) размера $length(MB)$, выбранный случайным образом

из обучающего набора данных. Обычно MB выбирается от нескольких десятков до

нескольких сотен обучающих примеров ,

в зависимости от размера обучающего набора данных,

∇L - градиент функции потерь $L(w_t; y_{it}, m(x_{it}))$ по параметрам модели m ,

x_{it} и y_{it} - i -й обучающий пример из mini-batch набора данных.

(3)

Модификации с инерцией

Градиентный спуск с инерцией (Momentum) и **Nesterov Accelerated Gradient (NAG)** являются расширениями стандартного градиентного спуска, которые позволяют учитывать различный масштаб признаков и быстрее продвигаться к оптимуму функции потерь со сложными линиями уровней.

Оптимизация с помощью Momentum - это метод стохастической оптимизации, который ускоряет процесс обучения за счет накопления «импульса» градиентов векторов.

В процессе обучения градиенты вычисляются на каждом шаге для обновления параметров модели. При использовании Momentum, градиенты на каждом шаге умножаются на коэффициент «импульса» (обычно примерно равный 0.9), который указывает, какую долю предыдущего шага нужно сохранить при вычислении градиентов на следующем шаге.

Таким образом, импульс помогает ускорить компоненты градиента в тех направлениях, в которых они сохраняются на протяжении нескольких последовательных шагов, и затормозить в направлениях, в которых градиенты меняются быстро. Это позволяет быстрее и более стабильно сходиться к локальному оптимуму и уменьшить вероятность застревания в локальном минимуме.

При использовании метода Momentum также возможно снижение количества осцилляций (взлетов и падений) при движении к минимуму функции потерь.

Метод Momentum можно применять совместно с различными алгоритмами оптимизации, такими как SGD (стохастический градиентный спуск), Adam и RMSprop.

Nesterov Accelerated Gradient (NAG) - это метод стохастической оптимизации, который является модификацией метода Momentum. В методе Momentum градиенты на каждом шаге умножаются на коэффициент «импульса» и складываются с предыдущим шагом для обновления параметров модели. Однако, в этом случае, имеется небольшая задержка в обновлении параметров, так как градиенты вычисляются на текущей позиции, а обновление параметров происходит на следующем шаге.

Метод Nesterov Accelerated Gradient (NAG) позволяет исправить эту проблему, предварительно заглянув в будущее и вычислив градиент на предполагаемой позиции в следующем шаге, а не на текущей позиции, перед вычислением импульса. Это позволяет получить более точный градиент, что в свою очередь уменьшает вероятность перепрыгивания через локальный оптимум при обновлении параметров.

Более конкретно, в методе NAG сначала вычисляются градиенты на текущей позиции, затем делается шаг в направлении, заданном градиентами на следующей позиции, и только затем вычисляются импульсы, которые накапливаются и добавляются к шагу.

Преимущества модификаций с инерцией:

- Они помогают избежать локальных минимумов и ускоряют сходимость.
- Они позволяют обучать более глубокие и сложные модели с большим количеством параметров.
- Они устойчивы к застреванию в локальных оптимумах и помогают более быстро достичь глобального минимума функции потерь.

Формулы модификаций:

- Моментум (Momentum). Эта модификация градиентного спуска учитывает предыдущие изменения вектора градиента при обновлении весов модели. Это позволяет ускорить

обучение и уменьшить вероятность застревания в локальных минимумах.

$$v_{t+1} = \eta v_t + \alpha_t \nabla L(w_t)$$

$$w_{t+1} = w_t - v_{t+1}$$

где v_t - «momentum», усредненное направление движения на шаге t ,

η - коэффициент момента, обычно выбирается около 0.9 для большинства задач,

α_t - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

∇L - градиент функции потерь $L(w_t)$ по параметрам модели m ,

w_t - вектор параметров модели на шаге t .

(4)

Формула градиентного спуска с моментом добавляет вектор скорости v_t , который является экспоненциальным сглаживанием градиента на предыдущих шагах. Это помогает ускорить процесс оптимизации и сгладить колебания, особенно на «шумных» данных.

- Nesterov Accelerated Gradient (NAG). Это модификация градиентного спуска, которая улучшает сходимость за счет добавления момента движения. В отличие от обычного градиентного спуска, который обновляет веса на основе текущего градиента, Nesterov momentum сначала делает шаг в направлении предыдущего накопленного градиента, а затем корректирует его на основе текущего градиента.

$$v_{t+1} = \eta v_t + \alpha_t \nabla L(w_t - \eta v_t)$$

$$w_{t+1} = w_t - v_{t+1}$$

где v_t - «Nesterov momentum», усредненное направление движения на шаге t ,

η - коэффициент момента, обычно выбирается около 0.9 для большинства задач,

α_t - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

∇L - градиент функции потерь $L(w_t)$ по параметрам модели m ,

w_t - вектор параметров модели на шаге t .

(5)

Формула Nesterov Accelerated Gradient добавляет корректировку вектора скорости v_t перед вычислением градиента. Это позволяет более точно оценить градиент cost функции и ускоряет процесс оптимизации.

Модификации с адаптивной скоростью изменения каждого веса

Модификации градиентного спуска с адаптивной (независимой для каждого веса) скоростью изменения весов, такие как Adagrad, RMSProp, являются расширениями стандартного градиентного спуска, которые позволяют эффективно оптимизировать функции потерь, учитывая различные характеристики градиента.

Adagrad - это метод стохастической оптимизации, который позволяет автоматически адаптировать скорость обучения для каждого параметра модели в процессе обучения.

Основная идея метода Adagrad заключается в том, чтобы присваивать больший вес параметрам, для которых наблюдается более редкое обновление, и меньший вес параметрам, для которых наблюдается частое обновление. Для этого на каждом шаге метод Adagrad вычисляет диагональную матрицу G , которая хранит сумму квадратов градиентов для каждого параметра на протяжении всего обучения. Затем скорость обучения для каждого параметра на текущем шаге вычисляется путем деления скорости обучения на корень из суммы квадратов градиентов для данного параметра на предыдущих шагах.

Преимущество метода Adagrad заключается в том, что он позволяет более эффективно обновлять параметры модели для различных направлений в пространстве параметров, что позволяет более быстро и точно сходиться к оптимальному решению. Кроме того, Adagrad не требует подбора скорости обучения, что является важным преимуществом в задачах с большим количеством параметров. Однако, метод Adagrad может иметь проблемы с сходимостью в случае, когда сумма квадратов градиентов становится очень большой. Эта проблема может быть решена с помощью методов оптимизации, основанных на Adagrad, таких как Adadelta и RMSprop.

RMSprop (Root Mean Square Propagation) - это метод стохастической оптимизации, который использует историю градиентов для адаптации скорости обучения на каждом шаге.

В методе RMSprop, вместо хранения суммы квадратов градиентов для каждого параметра, как в Adagrad, используется экспоненциально взвешенное среднее (exponential moving average) для хранения истории градиентов.

Преимущество метода RMSprop заключается в том, что он позволяет более эффективно обновлять параметры модели для различных направлений в пространстве параметров, учитывая историю градиентов. Кроме того, RMSprop автоматически адаптирует скорость обучения для каждого параметра, что упрощает настройку гиперпараметров.

Недостатком метода RMSprop является то, что он не учитывает адаптацию скорости обучения в различных направлениях в пространстве параметров, что может привести к неэффективному движению вдоль осей, где градиенты меньше, чем вдоль осей, где градиенты больше.

Преимущества градиентного спуска с адаптивной скоростью изменения каждого веса

- Они позволяют быстрее и более эффективно оптимизировать функции потерь, учитывая различные характеристики градиента.
- Они могут помочь избежать застревания в локальных оптимумах и быстрее достигать глобального минимума функции потерь.
- Они устойчивы к изменениям в данных и помогают улучшить обобщающую способность модели.

Формулы модификаций:

- Адаптивный градиентный спуск (Adaptive GD, AdaGrad). Эта модификация алгоритма позволяет изменять длину шага (learning rate) в зависимости от геометрических свойств поверхности функции потерь. Например, метод Adagrad позволяет изменять длину шага для каждого параметра модели в зависимости от того, как часто он обновляется.

$$\begin{aligned} G_{jt+1} &= G_{jt} + \nabla L(w_t)_j^2 \\ w_{jt+1} &= w_{jt} - \frac{\alpha_t}{\sqrt{G_{jt} + \epsilon}} \nabla L(w_t)_j \end{aligned}$$

G_{jt} - накопленные квадраты j компонента градиента для шага t ,

∇L - градиент функции потерь $L(w_t)$ по параметрам модели t , (6)

α_t - скорость обучения (learning rate),

определяющая величину шага в каждой итерации,

ϵ - значение, добавляемое для стабилизации вычислений,

w_t - вектор параметров модели на шаге t .

Формула Adagrad предлагает уникальный подход к выбору скорости обучения для каждого параметра, основанный на адаптивном масштабировании скорости обучения в соответствии с суммой квадратов предыдущих градиентов. Adagrad эффективен для работы с разреженными данными и шумными градиентами, что делает его популярным в обработке естественного языка и компьютерном зрении.

- Алгоритм RMSprop (Root Mean Square Propagation). Эта модификация алгоритма корректирует скорость обучения каждого параметра в сети на основе среднего квадрата градиентов для этого параметра. Интуиция за алгоритмом RMSprop заключается в том, чтобы уделять больше внимания последним градиентам и меньше внимания прошлым градиентам при обновлении параметров модели.

$$\begin{aligned} G_{jt+1} &= \eta G_{jt} + (1 - \eta) \nabla L(w_t)_j^2 \\ w_{jt+1} &= w_{jt} - \frac{\alpha_t}{\sqrt{G_{jt} + \epsilon}} \nabla L(w_t)_j \end{aligned}$$

G_{jt} - накопленные квадраты j компонента градиента для шага t ,

η - коэффициент затухания (damping factor),

обычно выбирается около 0.9 для большинства задач,

∇L - градиент функции потерь $L(w_t)$ по параметрам модели t ,

α_t - скорость обучения (learning rate),

определяющая величину шага в каждой итерации,

ϵ - значение, добавляемое для стабилизации вычислений,

w_t - вектор параметров модели на шаге t .

Формула RMSprop использует скользящее среднее квадрата градиента, чтобы нормировать скорость обучения для каждого параметра. Это позволяет уменьшить влияние выбросов в градиентах на процесс обучения и улучшить сходимость. Коэффициент затухания позволяет «забывать» более старые значения градиента, чтобы учитывать только более актуальные значения.

Модификации с инерцией и адаптивной скоростью изменения каждого веса

Adam (Adaptive Moment Estimation) - это метод стохастической оптимизации, который комбинирует идеи методов Momentum и RMSprop. Adam использует экспоненциально взвешенное среднее градиентов, как в RMSprop, и добавляет коррекцию смещения, как в методе Momentum, чтобы обеспечить более стабильную и сбалансированную оптимизацию.

Преимущества модификации с инерцией и адаптивной скоростью изменения каждого веса:

- Адаптивная скорость обучения: Adam автоматически адаптирует скорость обучения для каждого параметра в процессе оптимизации, что позволяет более быстро достигнуть оптимального значения функции потерь.
- Контроль момента: Adam использует два экспоненциально слаженных момента градиента для адаптивного выбора скорости обучения. Это позволяет более эффективно обрабатывать шум в данных и делать более точные оценки градиента.
- Эффективность: Adam показывает хорошую производительность на больших наборах данных и быстро сходится к оптимальному значению функции потерь.

- Использование памяти: Adam хранит только первые и вторые моменты градиента, что позволяет оптимизировать параметры на больших наборах данных, не требуя больших объемов оперативной памяти.
- Сходимость: Adam показывает быструю сходимость на многих задачах, и его параметры могут быть настроены автоматически при помощи алгоритмов оптимизации гиперпараметров.

Недостатком метода Adam является его сложность, которая может замедлить процесс обучения на больших моделях. Также в некоторых случаях метод Adam может не сходиться к оптимальному решению. Для некоторых задач может быть более эффективным использовать методы оптимизации, основанные на методе Momentum или RMSprop.

Формула модификации:

- Формула Adam объединяет идеи из Momentum и RMSprop и использует два скользящих средних - скользящее среднее градиента m_t и скользящее среднее квадрата градиента v_t . Она исправляет начальное смещение в начале оптимизации и адаптивно регулирует скорость обучения для каждого параметра. Adam является одним из наиболее эффективных методов градиентного спуска и часто используется в глубоком обучении.

$$\begin{aligned} g_t &= \nabla L(w_t) \\ m_t &= \beta_1 m_{t-1} + (1 - \beta_1)g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2)g_t^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\ w_{t+1} &= w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \end{aligned}$$

где g_t - градиент на шаге t ,

m_t - скользящее среднее градиента на шаге t ,

β_1 и β_2 - коэффициенты момента,

обычно выбираются около 0.9 и 0.999 соответственно,

v_t - скользящее среднее квадрата градиента на шаге t ,

\hat{m}_t и \hat{v}_t - исправленные значения скользящего среднего градиента

и квадрата градиента на шаге t , чтобы учитывать начальное смещение в начале оптимизации,

v_t - скользящее среднее квадрата градиента на шаге t ,

α_t - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

ϵ - значение, добавляемое для стабилизации вычислений,

w_t - вектор параметров модели на шаге t .

(8)

0.1.6 Вопросы для самопроверки

Чем отличается loss function от cost function в контексте машинного обучения?

1. Loss function используется для оценки ошибки на одном примере, а cost function - для оценки ошибки на всем наборе данных.

2. Loss function используется для оценки ошибки на всем наборе данных, а cost function - для оценки ошибки на одном примере.
3. Loss function и cost function - это одно и то же понятие.
4. Loss function и cost function отличаются только в своем названии и используются для оценки ошибки на всем наборе данных.

Правильные ответы: 1

Что происходит на каждой итерации алгоритма градиентного спуска?

1. Вычисляется значение функции потерь на одном случайно выбранном примере данных, затем значения параметров модели обновляются для уменьшения этого значения функции потерь.
2. Вычисляются частные производные функции потерь по всем параметрам модели, затем значения параметров обновляются с использованием этих частных производных.
3. Вычисляется производная функции потерь по одному случайно выбранному параметру модели, затем значение этого параметра обновляется с использованием этой производной.
4. Вычисляется значение функции потерь на всем наборе данных, затем значения параметров модели обновляются для уменьшения этого значения функции потерь.

Правильные ответы: 2

В каких модификациях градиентного спуска обновление весов происходит адаптивно для каждого из весов, с учетом особенностей признаков (3 ответа)?

1. Градиентный спуск с инерцией
2. Nesterov Accelerated Gradient
3. Adagrad
4. RMSProp
5. Adam

Правильные ответы: 3,4,5

0.1.7 Резюме по разделу

Функции потерь и функционалы ошибок являются важными инструментами в машинном обучении, так как они позволяют оценивать качество модели и управлять процессом обучения.

Функция потерь - это функция, которая оценивает расхождение между предсказанными значениями модели и фактическими значениями целевой переменной на конкретном примере данных. Функционал ошибки - это функция, которая оценивает общее качество модели на всем наборе данных. Функционал ошибки вычисляется путем усреднения функции потерь по всем примерам в наборе данных. Оптимизация функционала ошибки является целью обучения модели, так как это позволяет создать модель, которая хорошо обобщает данные и способна делать точные прогнозы на новых данных.

Функции потерь и функционалы ошибок используются для настройки параметров модели в процессе обучения. Цель состоит в том, чтобы минимизировать значение функции потерь

или функционала ошибки путем обновления параметров модели. Оптимизация функции потерь происходит с помощью градиентного спуска.

Существуют различные модификации алгоритма градиентного спуска:

- Модификации с обучением не по всей выборке (SGD, Mini-batch GD)
- Модификации с инерцией (Градиентный спуск с инерцией, Nesterov Accelerated Gradient)
- Модификации с адаптивной скоростью изменения каждого веса (Adagrad, RMSProp)
- Модификации с инерцией и адаптивной скоростью изменения каждого веса (Adam)

0.2 Регуляризация

Мы уже немного затрагивали тему регуляризации в предыдущем модуле. На этом занятии мы более подробно разберем различные методы регуляризации.

Регуляризация в машинном обучении — это методика, которая помогает снизить переобучение моделей. Переобучение возникает, когда модель слишком хорошо подстраивается под тренировочные данные, что приводит к тому, что она плохо обобщается на новые данные. Регуляризация может быть полезна в многих задачах машинного обучения, особенно в случаях, когда у вас мало тренировочных данных или когда модель имеет очень много параметров.

0.2.1 Основные методы регуляризации

Существует несколько основных методов регуляризации в машинном обучении, включая:

- L1-регуляризация (Lasso): добавляет к функции потерь модели сумму абсолютных значений весов, что приводит к разреженности весов и выбору наиболее значимых признаков.
- L2-регуляризация (Ridge): добавляет к функции потерь модели квадрат суммы всех весов, что приводит к сокращению весов и снижению влияния шумовых признаков.
- Elastic Net: комбинация L1-регуляризации и L2-регуляризации, которая помогает учесть преимущества обоих методов и уменьшить их недостатки.
- Dropout: случайным образом исключает некоторые узлы из обучения на каждом шаге, что помогает снизить переобучение и повысить обобщающую способность модели (используется в глубинном обучении).
- Data augmentation: методика, которая заключается в создании новых тренировочных данных путем искажения и изменения исходных данных, что помогает снизить переобучение и увеличить обобщающую способность модели.
- Early stopping: методика, которая заключается в прекращении обучения модели, когда ее производительность на проверочном наборе данных перестает улучшаться, что помогает избежать переобучения и выбрать наилучшую модель.

L1-регуляризация, L2-регуляризация и Elastic Net можно использовать с любыми моделями, где оптимизируются веса с помощью функции потерь. Early stopping применяется в случаях, когда происходит мониторинг метрик качества или изменения норм весов. Dropout и Data augmentation применяются в основном в нейросетевых моделях.

0.2.2 Методы регуляризации, основанные на добавлении слагаемого к функции потерь модели

Давайте подробнее разберем те типы регуляризации, которые работают путем добавления дополнительного слагаемого к функции потерь модели. Существует два основных типа такой регуляризации: L1-регуляризация и L2-регуляризация. L1-регуляризация добавляет к функции потерь модели сумму абсолютных значений всех весов модели, тогда как L2-регуляризация добавляет к функции потерь модели квадрат суммы всех весов модели. Это дополнительное слагаемое, которое называется регуляризатором, заставляет модель предпочитать более простые решения, которые не зависят слишком сильно от конкретных тренировочных данных. Интуиция за данным действием следующая: большие веса признака свидетельствуют о том, что модель переобучилась. Соответственно, добавление дополнительного

слагаемого к функции потерь заставляет модель снижать значение весов признаков. Принцип регуляризации, основанной на добавлении слагаемого к функции потерь модели, можно продемонстрировать на следующей иллюстрации: с помощью полиномиальной функции 1, 4, и 8 степени мы пытаемся выучить закономерности в данных (Рисунок 8).

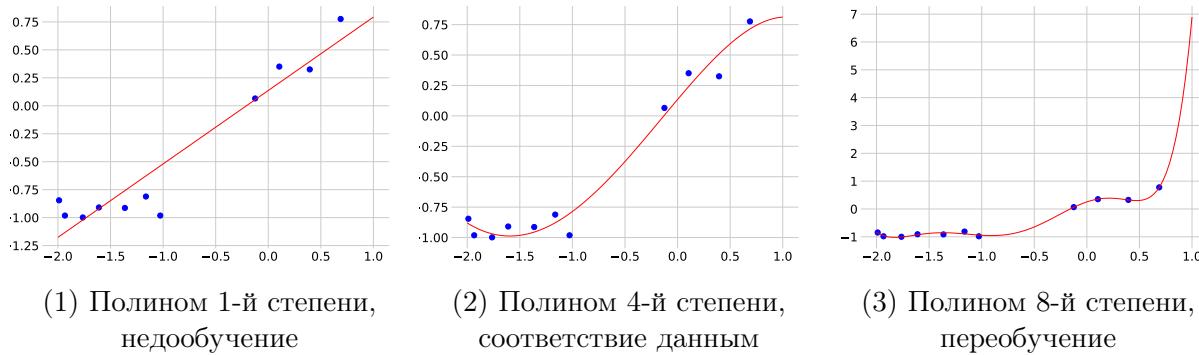


Рисунок 8: Иллюстрация обобщающей способности модели машинного обучения

Мы видим, что отсутствие регуляризации (Рисунок 8.3) приводит к слишком сильному подстраиванию под данные, что соответствует ситуации переобучения. Наоборот, слишком сильная регуляризация (Рисунок 8.1) приводит к упрощению модели и недообучению. Оптимальный вариант показан на рисунке 8.2. В этом случае регуляризация оптимальна, модель правильно улавливает закономерности в данных. Сила регуляризации в L1 и L2 регуляризации контролируется через параметр λ .

L1-регуляризация

L1-регуляризация (или Lasso, от англ. Least Absolute Shrinkage and Selection Operator) - это метод регуляризации в машинном обучении, который использует сумму абсолютных значений весов модели, чтобы ограничить их величину. Он работает путем добавления слагаемого, пропорционального сумме абсолютных значений весов модели, к общей функции потерь модели (Уравнение 9).

$$L_{L1} = L + \lambda \sum_{j=1}^d |w_j|, \quad (9)$$

где L - функция потерь без регуляризации, λ - гиперпараметр, который контролирует силу регуляризации, d - количество признаков, w_j - вес признака j .

Плюсы L1-регуляризации:

- Одним из главных преимуществ L1-регуляризации является ее способность к отбору признаков: модель, обученная с использованием L1-регуляризации, часто имеет многие веса, которые равны нулю, что позволяет определить, какие признаки важны для предсказания и какие можно исключить. Это упрощает модель и уменьшает шум в данных, что может привести к более точным прогнозам.
- L1-регуляризация работает лучше, когда у модели есть несколько важных признаков и много незначительных признаков.

- L1-регуляризация может быть реализована с помощью различных методов оптимизации, включая стохастический градиентный спуск и координатный спуск.

Минусы L1-регуляризации:

- L1-регуляризация не работает хорошо, когда все признаки важны: она может уменьшить слишком много весов и потерять информацию.
- L1-регуляризация может быть менее эффективной, чем L2-регуляризация, если признаки коррелируют друг с другом, потому что она не может различать между ними.
- L1-регуляризация может быть более сложной для оптимизации, чем L2-регуляризация, потому что функция потерь не является дифференцируемой в точках, где весы равны нулю.

L2-регуляризация

L2-регуляризация (или Ridge) - это метод регуляризации в машинном обучении, который использует квадратичную сумму весов модели, чтобы ограничить их величину. Он работает путем добавления слагаемого, пропорционального сумме квадратов весов модели, к общей функции потерь модели (Уравнение 10).

$$L_{L2} = L + \lambda \sum_{j=1}^d w_j^2, \quad (10)$$

где L - функция потерь без регуляризации, λ - гиперпараметр, который контролирует силу регуляризации, d - количество признаков, w_j - вес признака j .

Плюсы L2-регуляризации:

- L2-регуляризация помогает уменьшить вариацию весов модели, уменьшить переобучение и улучшить обобщающую способность модели.
- L2-регуляризация может быть лучше, чем L1-регуляризация, когда все признаки важны для модели, или когда признаки сильно коррелируют друг с другом.
- Функция потерь с L2-регуляризацией имеет гладкую форму и может быть легче оптимизирована с помощью градиентных методов.

Минусы L2-регуляризации:

- L2-регуляризация не уменьшает веса до нуля, поэтому она не может использоваться для отбора признаков.
- L2-регуляризация может быть менее эффективной, чем L1-регуляризация, когда модель имеет много незначительных признаков, которые можно исключить из модели.
- L2-регуляризация неспособна различать между сильно коррелирующими признаками, поэтому она может сохранить все такие признаки в модели, в то время как L1-регуляризация может установить некоторые из них в ноль.

Отбор признаков с помощью L1-регуляризации

L1-регуляризация приводит к сжатию некоторых весов до нуля, что позволяет отбросить ненужные признаки и уменьшить размерность модели. Это происходит из-за свойств абсолютной функции, которая имеет резкий градиент в нуле. При увеличении коэффициента регуляризации λ модель становится более склонной к сокращению весов признаков до нуля.

Таким образом, L1-регуляризация является эффективным методом отбора признаков, так как она устанавливает нулевые веса для ненужных признаков, в результате чего они не влияют на предсказание модели. Однако, если все признаки являются важными для модели, L1-регуляризация может привести к потере точности, поскольку важные признаки также могут быть установлены в ноль.

Геометрическое объяснение, почему L1-регуляризация приводит к отбору признаков, связано с формой ограничений, наложенных на модель L1-регуляризацией.

Представим, что мы имеем двумерный набор данных с двумя признаками, и мы хотим создать модель регрессии для этого набора данных. Рассмотрим пространство параметров нашей модели, т.е. все возможные комбинации весов признаков. Это пространство параметров представляет собой двумерную плоскость. Если мы не используем регуляризацию, то модель может выбрать любую точку в этом пространстве параметров.

Когда мы применяем L1-регуляризацию, мы добавляем к функции потерь ограничение на сумму абсолютных значений весов модели. Функция ограничений имеет форму ромба с вершинами, соответствующими значениям, равным $\pm\lambda$ по каждой координате.

Когда мы применяем L2-регуляризацию, мы добавляем к функции потерь ограничение на сумму квадратов значений весов модели. Функция ограничений имеет форму круга с радиусом, равным λ .

Оптимизация функции потерь с L1-регуляризацией эквивалентна выбору точки на плоскости, которая пересекается с ромбом. Эта точка находится на одной из вершин ромба, что соответствует некоторой комбинации признаков, для которых веса равны нулю. В ситуации же с L2-регуляризацией гораздо меньше шансов, что линия уровня коснется пространства ограничений в точке пересечения с одной из осей (Рисунок 9). Таким образом, при использовании L1-регуляризации приводит к отбору признаков путем установления нулевых весов для ненужных признаков.

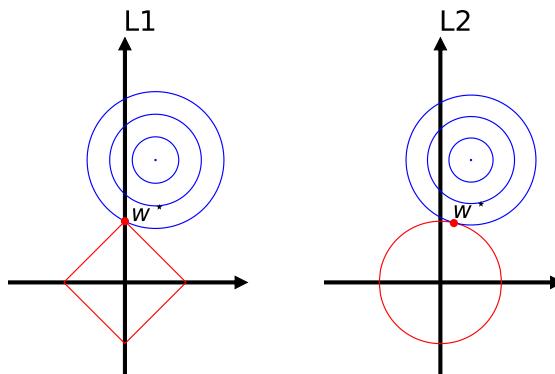


Рисунок 9: Отбор признаков с L1-регуляризацией

Такое геометрическое объяснение можно обобщить на пространства параметров большей размерности и на модели с несколькими признаками. Когда мы добавляем L1-регуляризацию в модель, ограничение на сумму абсолютных значений весов формирует многогранник, у которого некоторые вершины соответствуют комбинациям признаков с нулевыми весами. Оптимизация функции потерь с L1-регуляризацией приводит к выбору точки на этом многограннике, что соответствует выбору определенного подмножества признаков, для которых веса не равны нулю.

0.2.3 Вопросы для самопроверки

Выберите все методы регуляризации, основанные на добавлении слагаемого к функции потерь модели?

1. L1-регуляризация (Lasso)
2. L2-регуляризация (Ridge)
3. Data augmentation
4. Early stopping
5. Elastic Net
6. Dropout

Правильные ответы: 1, 2, 5

Какой из следующих признаков является признаком переобучения модели машинного обучения?

1. Низкая точность на тренировочных данных, но высокая точность на тестовых данных.
2. Высокая точность на тренировочных данных, но низкая точность на тестовых данных.
3. Равномерно высокая точность на тренировочных и тестовых данных.
4. Большое количество признаков в модели.
5. Малое количество эпох обучения модели.

Правильные ответы: 2

С помощью какого метода регуляризации можно отбирать важные признаки?

1. L1-регуляризация (Lasso)
2. L2-регуляризация (Ridge)
3. Data augmentation
4. Early stopping
5. Elastic Net
6. Dropout

Правильные ответы: 1

0.2.4 Резюме по разделу

Существует несколько основных методов регуляризации в машинном обучении, включая L1-регуляризацию (Lasso), L2-регуляризацию (Ridge), Elastic Net, Dropout, Data augmentation, Early stopping. Методы регуляризации, основанные на добавлении слагаемого к функции потерь модели - это L1-регуляризация (Lasso), L2-регуляризация (Ridge), Elastic Net (комбинация L1 и L2). С помощью L1 регуляризации можно отбирать важные признаки.

0.3 Линейная регрессия

Цель занятия: ученик может применить алгоритм линейной регрессии для решения задач регрессии на подготовленных и неподготовленных данных.

План занятия:

1. Визуальная демонстрация алгоритма
2. Описание алгоритма
3. Подготовка данных для алгоритма
4. Процесс обучения
5. Оценка качества алгоритма
6. Интерпретация признаков с помощью алгоритма
7. Модификации алгоритма
8. Область применения алгоритма
9. Плюсы и минусы алгоритма
10. Реализация алгоритма в Python

0.3.1 Визуальная демонстрация алгоритма

На практике достаточно часто встречаются ситуации линейной зависимости одной переменной от другой или других переменных. Линейную зависимость между двумя переменными можно вывести с помощью стандартных статистических методов, таких как корелляция. Если же целевая переменная зависит от нескольких признаков (в статистике их называют предикторами), используют более сложные модели. И первой из таких моделей является алгоритм линейной регрессии.

Визуальная демонстрация алгоритма представлена на иллюстрации 10.

Давайте разберемся с тем, что изображено на иллюстрации 10. На рисунке 10.1 приведена зависимость тормозного пути автомобиля от его скорости как пример построения линейной регрессии с одним признаком. Такая зависимость аппроксимируется линией и может быть визуализирована в двумерном пространстве. Алгоритм начинает поиск оптимального решения со случайных значений (синяя линия) и постепенно приближается к оптимальному значению (красная линия) (Рисунок 10.2). Если в наборе данных содержится два признака, то оптимальные значения будут аппроксимироваться плоскостью в трехмерном пространстве (Рисунок 10.3).

В линейной регрессии мы вычисляем ошибку модели как расстояние между каждой точкой данных и гиперплоскостью, которая является аппроксимацией линейной связи между независимыми и зависимой переменными. Это расстояние измеряется вдоль нормали (или перпендикуляра) к гиперплоскости.

Можно представить себе, что мы опускаем нормаль от каждой точки данных до гиперплоскости, получая расстояние между точкой данных и гиперплоскостью. Это расстояние называется остаточной суммой (residual sum) и является мерой того, насколько хорошо модель соответствует данным. В линейной регрессии мы стремимся минимизировать остаточную сумму, настраивая коэффициенты гиперплоскости таким образом, чтобы она была максимально близка к данным.

Для модели с d признаками (исключая свободный коэффициент), линейная регрессия аппроксимируется гиперплоскостью в $d+1$ мерном пространстве. Каждый признак умножается

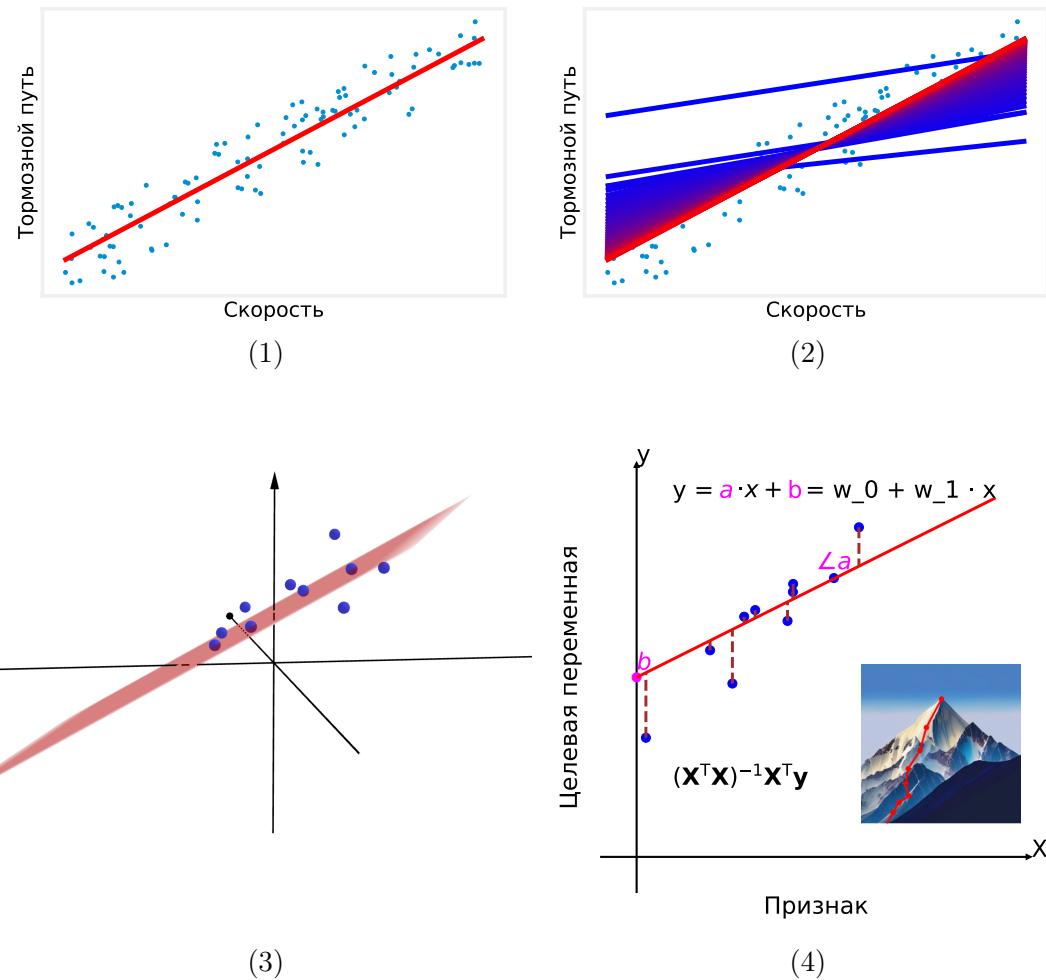


Рисунок 10: Визуальная демонстрация алгоритма линейной регрессии

на соответствующий вес. Также в модель добавляется настраиваемый свободный коэффициент (b или w_0), который позволяет модели не выдавать нулевой прогноз при нулевых значениях для остальных признаков. Для свободного коэффициента обычно добавляется «фиктивный признак», все значения которого равны 1. Предсказание модели в таком случае - это скалярное произведение вектора весов на вектор признаков. Рисунок 10.4) демонстрирует математическую суть линейной регрессии: в двумерном случае линейная регрессия задается хорошо известным еще со школы уравнением прямой (Уравнение 11).

$$y = a \cdot x + b,$$

где y - целевая (зависимая) переменная,

x - признак (независимая переменная),

a - угол наклона прямой (slope),

b - свободный коэффициент (intercept или bias), точка пересечения с осью y .

В терминах машинного обучения коэффициенты b и a обозначаются как w_1 и w_2 соответственно (веса модели). Для нахождения коэффициентов b и a можно пользоваться различными

методами: аналитическим (Формула $(X^T X)^{-1} X^T y$, не всегда применима) или градиентным спуском.

0.3.2 Описание алгоритма

Перейдем от визуального описания алгоритма линейной регрессии к примеру его использования и формальному определению.

Линейная регрессия с несколькими признаками, также называемая множественной линейной регрессией, является методом анализа данных, который позволяет определить линейную зависимость между целевой (зависимой) переменной и несколькими признаками (независимыми переменными).

Например, предположим, что вы хотите определить, какие факторы влияют на цену на недвижимость. В этом случае, зависимой переменной является цена на недвижимость, а независимыми переменными могут быть такие факторы, как площадь квартиры, количество комнат, расстояние до ближайшего метро, возраст здания и т.д.

Множественная линейная регрессия позволяет определить, как каждый из этих факторов влияет на цену недвижимости, и как их комбинация может объяснить изменения цен на рынке недвижимости.

Например, если мы построим модель множественной линейной регрессии для цен на недвижимость, используя площадь квартиры, количество комнат и расстояние до ближайшего метро как независимые переменные, то мы можем получить следующее уравнение:

$$\text{Цена} = 5000 + 100 * \text{Площадь} + 2000 * \text{Комнаты} - 500 * \text{Расстояние}$$

В этом уравнении коэффициент перед каждой независимой переменной (площадью, количеством комнат и расстоянием до ближайшего метро) показывает, как каждая из этих переменных влияет на цену на недвижимость. Например, увеличение площади квартиры на 1 кв. метр приведет к увеличению цены на 100 единиц, увеличение количества комнат на 1 приведет к увеличению цены на 2000 единиц, а увеличение расстояния до ближайшего метро на 1 км приведет к уменьшению цены на 500 единиц.

Таким образом, множественная линейная регрессия позволяет определить, как каждый из факторов влияет на зависимую переменную, а также как их комбинация влияет на изменения цен на рынке недвижимости.

Дадим формальное определение линейной регрессии.

Линейная регрессия - это метод машинного обучения, который используется для оценки линейной зависимости между набором признаков (независимых переменных) и целевой (зависимой) переменной. Линейная регрессия позволяет найти линейную функцию, которая наилучшим образом описывает зависимость между признаками и целевой переменной.

Математически линейная регрессия определяется следующим образом:

Пусть \mathbf{X} - матрица признаков размерности $n \times d$, где каждая строка представляет собой один наблюдаемый пример, а каждый столбец - один признак. Пусть \mathbf{y} - вектор целевых переменных размерности $n \times 1$. Тогда модель линейной регрессии выглядит следующим образом (Уравнение 12).

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + \mathbf{b},$$

где \mathbf{w} - вектор весов размерности $n \times 1$, \mathbf{b} - смещение (bias), (12)

$\hat{\mathbf{y}}$ - вектор предсказанных значений целевой переменной.

Если для \mathbf{b} добавить фиктивный признак, равный 1 для всех примеров, то уравнение примет еще более простой вид произведения матрицы признаков на вектор весов (Уравнение 13).

$$\hat{\mathbf{y}} = \mathbf{X} \cdot \mathbf{w} \quad (13)$$

Задача линейной регрессии заключается в нахождении оптимальных значений вектора весов \mathbf{w} и смещения \mathbf{b} таким образом, чтобы минимизировать ошибку (в примере дана среднеквадратичная ошибка) между предсказанными и реальными значениями целевой переменной (Уравнение 14):

$$\min_{w,b} \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

где \hat{y}_i - предсказанное значение целевой переменной для i -го примера, (14)

y_i - реальное значение целевой переменной для i -го примера, w - вектор весов,

b - смещение (bias), а m - количество примеров в обучающем наборе.

0.3.3 Подготовка данных для алгоритма

Перед применением алгоритма линейной регрессии необходимо выполнить следующие шаги подготовки данных:

- **Очистка данных:** удаление неполных или некорректных записей, заполнение пропущенных значений и преобразование данных в формат, подходящий для анализа.
- **Нормализация данных:** приведение данных к общему масштабу, чтобы каждый признак имел одинаковый вклад в анализ. Нормирование признаков является важным шагом в предобработке данных перед использованием линейной регрессии. Это связано с тем, что линейная регрессия оценивает веса (коэффициенты) каждого признака в модели, чтобы предсказать целевую переменную. Если масштабы разных признаков значительно отличаются друг от друга, то оценка весов может быть смещена в сторону признаков с большими значениями. Это может привести к неверным выводам и плохому качеству предсказания. Обычно используются два основных способа нормирования:
 - Стандартизация: признаки масштабируются так, чтобы их среднее значение было равно 0, а стандартное отклонение было равно 1.
 - Нормализация: признаки масштабируются так, чтобы их значения находились в диапазоне от 0 до 1.

После нормирования признаков, каждый из них будет иметь примерно одинаковый вклад в предсказание целевой переменной, что позволит линейной регрессии более точно оценивать их веса и делать более точные прогнозы.

- **Разбиение данных на обучающую и тестовую выборки:** обучающая выборка используется для обучения модели, а тестовая выборка - для оценки ее точности.

0.3.4 Процесс обучения

Для решения этой задачи используются различные методы, такие как аналитическое решение и метод градиентного спуска. Результатом обучения модели линейной регрессии является набор оптимальных значений весов \mathbf{w} и смещения \mathbf{b} , которые могут быть использованы для предсказания целевой переменной для новых наблюдений.

Аналитическое решение линейной регрессии

Существует аналитическое решение для линейной регрессии (Уравнение 15).

$$w^* = (X^T X)^{-1} X^T y,$$

где X — матрица объекты-признаки, y - вектор ответов для объектов, (15)

w^* — вектор оптимальных весов.

Давайте рассмотрим, как оно получено. Для вывода нормального уравнения линейной регрессии рассмотрим многомерную линейную регрессионную модель (Уравнение 16).

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id} + \epsilon_i, i = 1, 2, \dots, n$$

где y_i - значение зависимой переменной для i -го наблюдения,

$x_{i1}, x_{i2}, \dots, x_{id}$ - значения d независимых переменных

(признаков) для i -го наблюдения, w_0, w_1, \dots, w_d - параметры регрессии,

ϵ_i - случайная ошибка или шум, которая имеет нулевое среднее и постоянную дисперсию. (16)

Для удобства обозначим X матрицей признаков размерности $n \times (d+1)$, в которой первый столбец заполнен единицами, и w вектором размерности $(d+1) \times 1$, состоящим из параметров регрессии:

$$X = \begin{bmatrix} 1 & w_{11} & w_{12} & \cdots & w_{1d} \\ 1 & w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_{n1} & w_{n2} & \cdots & w_{nd} \end{bmatrix}, w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

Тогда модель может быть записана в матричной форме:

$$y = X \cdot w + \epsilon$$

Для нахождения оценок коэффициентов регрессии \hat{w} , которые минимизируют сумму квадратов ошибок, воспользуемся методом наименьших квадратов. Сначала нам нужно определить функцию ошибок $L(w)$, которая является суммой квадратов отклонений прогнозируемых значений от фактических значений:

$$L(w) = \sum_{i=1}^n (y_i - w_0 - w_1 x_{i1} - w_2 x_{i2} - \cdots - w_d x_{id})^2$$

Для минимизации функции $L(w)$ найдем ее производные по каждому элементу вектора w и приравняем их к нулю:

$$\frac{\partial L(w)}{\partial w} = 2X^T(Xw - y) = 0,$$

где $\frac{\partial L(w)}{\partial w}$ - частные производные $L(w)$ по весам,

X^T - транспонированная матрица X , а y - вектор значений целевой переменной.

Умножив обе части уравнения на X^T , получим:

$$X^T X w = X^T y$$

Это называется нормальным уравнением линейной регрессии. Решив его относительно вектора параметров w , получим:

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

Таким образом, мы получили аналитическую формулу для оценки коэффициентов линейной регрессии. Оценки параметров w^* , которые минимизируют сумму квадратов ошибок, находятся как решение нормального уравнения.

Уравнение $w = (X^T X)^{-1} X^T y$, которое также называется «нормальным уравнением», очень часто не применимо на практике в силу следующих причин:

- Большой размер матрицы X : на обращение матрицы требуется порядка n^3 операций
- Несуществование обратной матрицы: Если матрица $(X^T X)$ не имеет обратной матрицы, то уравнение не может быть использовано. Это может произойти, например, когда столбцы матрицы линейно зависимы.
- Некорректность модели: Если модель линейной регрессии неправильно специфицирована, то уравнение не даст правильных результатов. Например, если зависимость между переменными является нелинейной, то линейная регрессия не будет работать правильно.
- Наличие выбросов: Если в данных есть выбросы, то уравнение может дать неверные результаты. Выбросы могут сильно влиять на оценку параметров модели и приводить к неправильным выводам.
- Недостаточный размер выборки: Если выборка слишком мала, то уравнение может давать неустойчивые оценки параметров модели. В таких случаях необходимо использовать более сложные методы оценки параметров.
- Нарушение предположений модели: Если предположения модели о распределении ошибок не соблюдаются, то уравнение может давать неверные результаты. Например, если ошибки не являются нормально распределенными или имеют гетероскедастичность, то уравнение может дать неправильные оценки параметров модели.

Линейная регрессия в матричной форме

На практике гораздо чаще применяются численные методы решения линейной регрессии, такие как градиентный спуск. Функция ошибки для линейной регрессии может быть записана так:

$$L(w) = \frac{1}{2n} * \sum_{i=1}^n (y_i - Xw)^2,$$

где y - вектор ответов, X - матрица объекты-признаки,

w - вектор параметров, n - количество наблюдений.

Градиентный спуск находит минимум функции ошибки, обновляя параметры модели на каждой итерации в направлении антиградиента функции ошибки:

$$w = w - \alpha \frac{1}{n} X^T (Xw - y),$$

где α - скорость обучения, которая контролирует ,

размер шага на каждой итерации.

Градиентный спуск повторяется до тех пор, пока значение функции ошибки не перестанет существенно изменяться или пока не будет достигнуто максимальное количество итераций.

Использование градиентного спуска для оптимизации линейной регрессии имеет некоторые преимущества, такие как возможность работать с большими объемами данных и возможность настройки гиперпараметров, таких как скорость обучения. Вот шаги обучения алгоритма линейной регрессии с помощью градиентного спуска в общем виде:

1. Инициализация: Выберите начальные значения для параметров модели линейной регрессии, таких как веса (коэффициенты) и смещение (intercept). Можно выбрать случайные значения или инициализировать нулями.
2. Определение функции потерь: Определите функцию потерь, которую вы хотите минимизировать. Для линейной регрессии часто используется среднеквадратичная ошибка (MSE), которая вычисляется как средняя квадратичная разница между предсказанными значениями и истинными значениями целевой переменной.
3. Вычисление градиента: Вычислите градиент функции потерь по каждому параметру модели. Градиент показывает направление наиболее быстрого возрастания функции потерь. Для линейной регрессии градиент можно вычислить аналитически.
4. Обновление параметров: Используя градиент и шаг обучения (learning rate), обновите значения параметров модели. Шаг обучения определяет размер шага, на который нужно изменить параметры модели в направлении, противоположном градиенту. Обновление параметров выполняется путем вычитания шага обучения, умноженного на градиент, от текущих значений параметров.
5. Повторение шагов: Повторяйте шаги 3 и 4 до достижения условия остановки. Условием остановки может быть достижение максимального числа итераций, достижение требуемой точности или сходимость функции потерь.
6. Возврат результата: Возвращение обновленных значений параметров модели линейной регрессии, которые представляют оптимальные значения для наилучшего соответствия данных.

На практике могут использоваться различные оптимизации и модификации алгоритма градиентного спуска для улучшения скорости и стабильности обучения, такие как регуляризация или мини-пакетный градиентный спуск.

0.3.5 Оценка качества алгоритма

Для оценки качества алгоритма линейной регрессии часто используют следующие метрики:

- **Mean Squared Error (MSE):** средняя квадратичная ошибка между прогнозами и истинными значениями.
- **Mean Absolute Error (MAE):** средняя абсолютная ошибка между прогнозами и истинными значениями.
- **Huber loss:** комбинация MSE и MAE, которая более устойчива к выбросам в данных, чем MSE

0.3.6 Интерпретация признаков с помощью алгоритма

Интерпретация признаков в линейной регрессии заключается в понимании того, как каждый признак влияет на зависимую переменную в модели. Это может помочь понять, какие признаки наиболее важны для прогнозирования целевой переменной и как они связаны с ней.

Наиболее простым способом оценить влияние того или иного признака - посмотреть на весовой коэффициент при признаке. Коэффициенты признаков в линейной регрессии представляют величину и направление влияния каждого признака на зависимую переменную. Если значение признака большое, можно сказать, что его значение также большое (при условии нормализованных данных). Если коэффициент положительный, то увеличение значения признака будет приводить к увеличению значения зависимой переменной. Если коэффициент отрицательный, то увеличение значения признака будет приводить к уменьшению значения зависимой переменной.

0.3.7 Модификации алгоритма

Существует несколько модификаций алгоритма линейной регрессии, которые могут быть применены в зависимости от задачи и данных. Некоторые из них:

- **Ridge regression (Гребневая регрессия):** добавление регуляризации L2 в функцию потерь для борьбы с переобучением. Это позволяет уменьшить веса модели, чтобы она была менее склонна к переобучению и более устойчива к шуму в данных.
- **Lasso regression (Лассо регрессия):** добавление регуляризации L1 в функцию потерь, чтобы стимулировать разреженность весов модели. Это означает, что некоторые веса будут установлены на ноль, что может помочь идентифицировать наиболее важные признаки.
- **Elastic Net:** комбинация L1 и L2 регуляризации в функции потерь. Это позволяет улучшить баланс между разреженностью и устойчивостью к шуму.
- **Polynomial regression (Полиномиальная регрессия):** расширение линейной регрессии путем добавления полиномиальных признаков для учета нелинейных зависимостей между признаками и целевой переменной.
- **Robust regression (Робастная регрессия):** модификация линейной регрессии, которая более устойчива к выбросам в данных. Она использует альтернативные функции потерь, такие как Huber loss, которые уменьшают вклад выбросов в обучении модели.

0.3.8 Область применения алгоритма

Линейная регрессия является одним из самых простых и наиболее распространенных методов машинного обучения, и может быть применена во многих областях, включая:

- **Финансовый анализ:** линейная регрессия может использоваться для прогнозирования цен акций, доходности инвестиций и других финансовых показателей.
- **Маркетинг:** линейная регрессия может использоваться для анализа данных о продажах и прогнозирования спроса на товары и услуги.
- **Медицинская статистика:** линейная регрессия может использоваться для анализа данных о заболеваемости и смертности, прогнозирования риска развития заболеваний и определения факторов, влияющих на здоровье.
- **Инженерия:** линейная регрессия может использоваться для анализа данных в различных областях, таких как электроника, механика и транспорт, для прогнозирования характеристик и поведения систем.
- **Социальные науки:** линейная регрессия может использоваться для анализа социальных данных, таких как опросы общественного мнения, для прогнозирования выборов, определения влияния социальных факторов на поведение людей и т.д.

В целом, линейная регрессия может быть использована для анализа любых данных, для которых есть зависимость между одной или несколькими признаками и целевой переменной. Однако, следует учитывать, что линейная регрессия может быть неэффективной для данных с нелинейными зависимостями и слабыми связями между переменными.

0.3.9 Плюсы и минусы алгоритма

Алгоритм линейной регрессии имеет ряд преимуществ и недостатков.

Плюсы:

- Простота реализации и понимания. Это один из самых простых алгоритмов машинного обучения.
- Может использоваться для задач регрессии и оценки влияния признаков.
- Эффективен на больших выборках и может быстро обучаться на большом количестве признаков.
- Хорошо интерпретируется и может помочь понять важность каждого признака.

Минусы:

- Чувствительность к выбросам и шуму в данных.
- Требует выполнения предположения о линейной зависимости между признаками и целевой переменной, что может быть проблематично в некоторых случаях.
- Не учитывает взаимодействие между признаками, что может привести к недооценке важности некоторых признаков.
- Может быть склонен к переобучению, если в модели слишком много признаков или коэффициенты признаков сильно отличаются друг от друга.

В целом, алгоритм линейной регрессии также имеет простую и понятную логику, но его эффективность может быть низкой в случае шумных данных и если не выполняются предположения о линейной зависимости между признаками и целевой переменной.

0.3.10 Реализация алгоритма в Python

В библиотеке Scikit-learn в модуле linear_model реализован класс **LinearRegression** для линейной регрессии.

Класс LinearRegression имеет следующие параметры:

- **fit_intercept**: булевый параметр, указывающий, должно ли применяться смещение (intercept/bias). По умолчанию равен True.
- **copy_X**: булевый параметр, указывающий, должны ли данные быть скопированы перед обучением модели. По умолчанию равен True.
- **n_jobs**: количество параллельных задач, которые используются для расчета. По умолчанию равен None, что означает использование всех доступных процессоров.

Класс LinearRegression имеет методы **fit(X, y)** для обучения модели на данных X и y, а также метод **predict(X)** для предсказания целевых значений для новых данных X. Кроме того, класс LinearRegression имеет методы **score(X, y)** и **get_params()** для получения коэффициента детерминации (R-квадрат) и параметров модели соответственно.

0.3.11 Вопросы для самопроверки

Как чаще всего происходит обучение алгоритма линейной регрессии?

1. алгоритм просто запоминает обучающую выборку, чтобы затем измерить расстояние от нового объекта до объектов в обучающей выборке
2. с помощью градиентного спуска
3. с помощью метода Ньютона

Правильные ответы: 2

Как вычисляется значение для нового объекта в задаче регрессии с помощью алгоритма линейной регрессии?

1. Значение вычисляется как среднее значение целевой переменной в обучающей выборке.
2. Значение вычисляется как скалярное произведение вектора признаков нового объекта на вектор весов модели.
3. Значение вычисляется как медианное значение целевой переменной в обучающей выборке.
4. Значение вычисляется как максимальное значение целевой переменной в обучающей выборке.

Правильные ответы: 2

Выберите плюсы алгоритма линейной регрессии:

1. Может быть склонен к переобучению, если в модели слишком много признаков или коэффициенты признаков сильно отличаются друг от друга.
2. Может использоваться для задач регрессии и оценки влияния признаков.
3. Не учитывает взаимодействие между признаками, что может привести к недооценке важности некоторых признаков.
4. Простота реализации и понимания. Это один из самых простых алгоритмов машинного обучения.
5. Требует выполнения предположения о линейной зависимости между признаками и целевой переменной, что может быть проблематично в некоторых случаях.
6. Хорошо интерпретируется и может помочь понять важность каждого признака.
7. Чувствительность к выбросам и шуму в данных.
8. Эффективен на больших выборках и может быстро обучаться на большом количестве признаков.

Правильные ответы: 2, 4, 6, 8

0.3.12 Резюме по разделу

Алгоритм линейной регрессии - это метод машинного обучения, который используется для аппроксимации линейной зависимости между независимыми и зависимыми переменными. Алгоритм предсказывает значение зависимой переменной для новых наблюдений, основываясь на линейной комбинации значений независимых переменных и коэффициентов регрессии.

Коэффициенты регрессии - это параметры модели, которые определяют веса независимых переменных в предсказании зависимой переменной. Чтобы определить коэффициенты, используются такие функционалы ошибок, как MSE, MAE и функционал ошибок Хьюбера.

Основными преимуществами линейной регрессии являются простота и интерпретируемость модели, а также возможность использования для прогнозирования непрерывных значений. Недостатками алгоритма являются чувствительность к выбросам и нелинейным зависимостям между переменными.

Также существуют модификации алгоритма, такие как регуляризованная линейная регрессия, которая позволяет уменьшить переобучение и улучшить обобщающую способность модели, а также линейная регрессия с использованием полиномиальных признаков, которая может улучшить точность модели при нелинейной зависимости между переменными.

0.4 Резюме по модулю

В этом модуле мы рассмотрели такие темы, как градиентный спуск для обучения дифференцируемых моделей машинного обучения, регуляризация моделей машинного обучения для борьбы с переобучением, а также познакомились с алгоритмом линейной регрессии.