GeekBrains

Функции потерь и оптимизация

Библиотеки Python для Data Science

Оглавление

Введение

В лекции о линейных моделях мы узнали, как использовать линейные алгоритмы для решения задачи классификации и регрессии. Во время лекций о линейной и логистической регрессии мы изучили, как построить модели для прогнозирования значений непрерывной и категориальной переменной соответственно.

Однако у нас есть много других алгоритмов машинного обучения, которые можно использовать для решения задачи прогнозирования. Но независимо от выбранного алгоритма, нам нужно как-то оценить его качество и настроить его параметры.

Это приводит нас к понятию функции потерь и оптимизации. Функция потерь — это метрика, которая измеряет, насколько хорошо модель предсказывает целевую переменную на обучающем наборе данных. Оптимизация — это процесс настройки параметров модели таким образом, чтобы минимизировать значение функции потерь и улучшить качество предсказаний модели.

На этой лекции вы найдете ответы на такие вопросы как / узнаете:

- Что такое функция потерь
- Функции потерь регрессии и классификации
- Узнаем, как обучается модель

Термины, используемые в лекции

Функция потерь — это метрика, которая измеряет, насколько хорошо модель предсказывает целевую переменную на обучающем наборе данных.

Оптимизация — это процесс настройки параметров модели таким образом, чтобы минимизировать значение функции потерь и улучшить качество предсказаний модели.

Гомоскедастичность — свойство, означающее постоянство условной дисперсии вектора или последовательности случайных величин.

Понятие функции потерь

Функция потерь — это инструмент, который позволяет оценить, насколько хорошо модель машинного обучения справляется с поставленной задачей.

Перед тем, как перейти к функциям потерь, давайте вспомним основные понятия машинного обучения. В процессе обучения модель получает на вход некоторый набор данных, и ее задача — извлечь из этого набора данных общие закономерности, чтобы в дальнейшем модель смогла делать предсказания на новых данных.

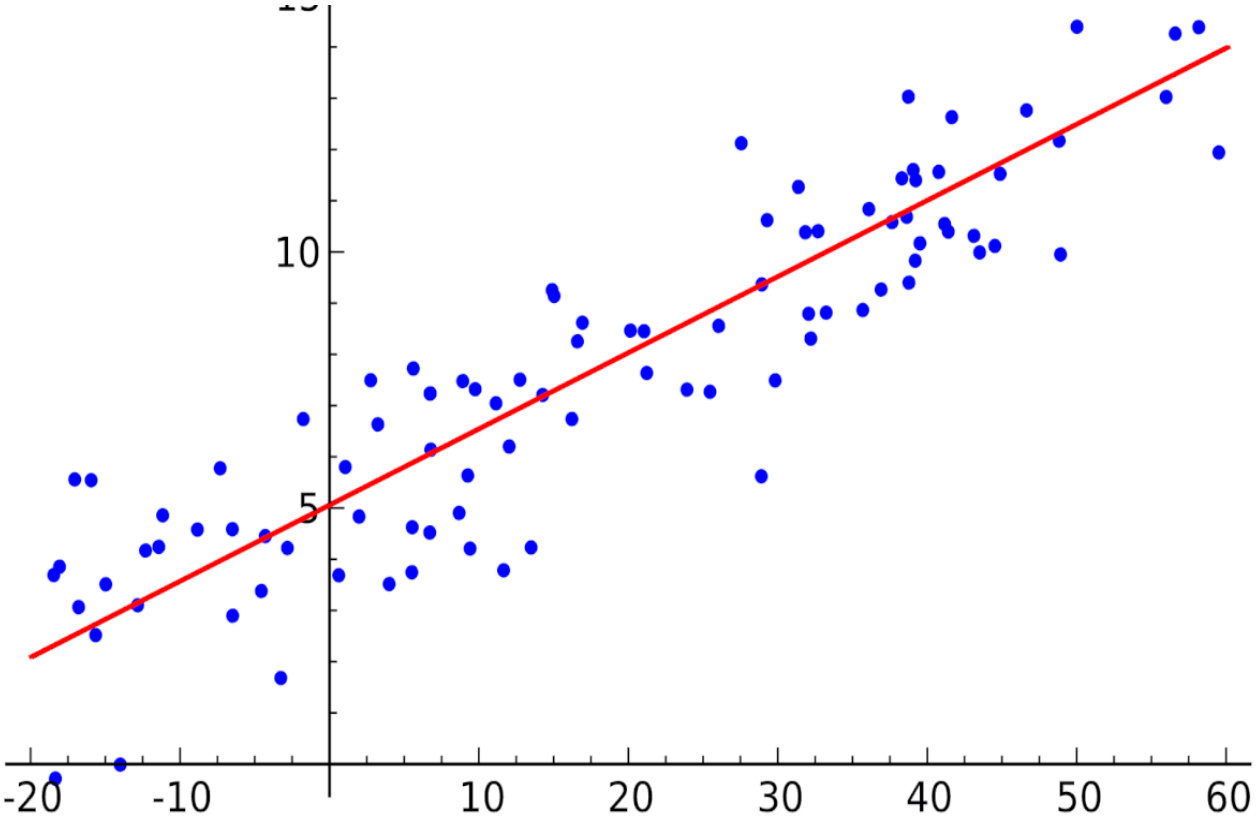
Однако модель ошибается в своих предсказаниях, и часто нам нужно иметь способ измерения этих ошибок. Вот тут-то нам и пригодятся функции потерь!

Функция потерь — это способ измерения ошибки модели. Она сравнивает предсказанные значения модели с фактическими значениями в данных и возвращает число, которое показывает насколько сильно модель ошибается. Чем меньше значение функции потерь, тем лучше модель справляется с поставленной задачей.

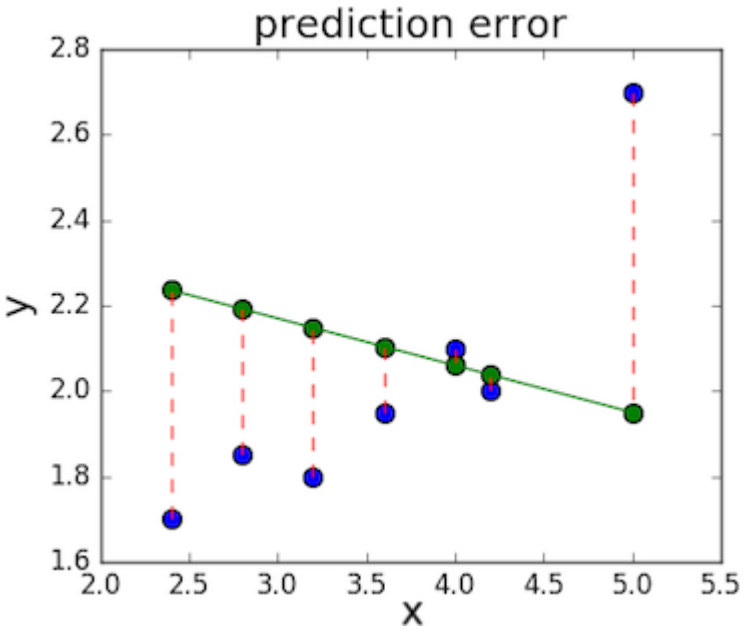
Важно отметить, что выбор функции потерь должен соответствовать задаче, которую мы пытаемся решить. Например, для задачи регрессии, когда мы предсказываем некоторую величину, часто используется функция потерь среднеквадратической ошибки (Mean Squared Error, MSE). Для задачи классификации, когда предсказываем принадлежность к классу, может быть использована функция потерь "логарифмическая функция потерь" (Log Loss).

Давайте вспомним, что из себя представляет задача линейной регрессии:
Линейная регрессия — это зависимость переменной x от одной или нескольких других переменных (факторов, регрессоров, независимых переменных), моделируемая линейной зависимой функцией.

Линейная регрессия относится к задаче определения "наилучшего соответствия прямой линии" по набору точек данных и является простым предшественником нелинейных методов, используемых для обучения нейронных сетей.



Функция потерь — это мера количества ошибок, допущенных линейной регрессией на наборе данных. Существует несколько функций потерь, каждая из которых рассчитывает расстояние между прогнозируемым и фактическим значениями y(x).



Поэтому перед тем, как выбрать функцию потерь, нужно четко представлять себе задачу и желаемый результат.

Очень распространенная функция потерь называется средней квадратичной ошибкой (MSE); чтобы вычислить MSE, возьмите все значения ошибок, посчитайте их квадраты и возьмите среднее значение. Она является одной из наиболее популярных функций потерь и обычно используется, когда ошибки в предсказаниях имеют одинаковую важность.

В первую очередь, необходимо задать модель зависимости объясняемой переменной y от объясняющих ее факторов, функция зависимости будет линейной:

$$y = w_0 + \sum_{i=1}^{m} w_i x_i$$

Если мы добавим фиктивную размерность $x_0 = 1$ для каждого наблюдения, тогда линейную форму можно переписать чуть более компактно, записав свободный член w_0 под сумму:

$$y = \sum_{i=0}^{m} w_i x_i = \vec{w}^T \vec{x}\$$

Если рассматривать матрицу наблюдения-признаки, у которой в строках находятся примеры из набора данных, то нам необходимо добавить единичную колонку слева. Зададим модель следующим образом:

$$\vec{y} = X\vec{w} + \epsilon,$$

где $\vec{y} \in \mathbb{R}^n\$$ – является объясняемой переменной;
w – вектором параметров модели (весами);
X – матрица наблюдений и признаков размерности n строк на m + 1 столбцов с полным рангом по столбцам: $\operatorname{rank}(X) = m + 1$;

ϵ – случайной переменной, представляющей случайную ошибку модели, которую нельзя предсказать.

Можно записать выражение для каждого отдельного наблюдения следующим образом:

$$y_i = \sum_{j=0}^{m} w_j X_{ij} + \epsilon_i$$

Также есть определенные ограничения на модель, иначе это будет другой тип регрессии, а не линейная:

1. Математическое ожидание случайных ошибок равно нулю
$\forall i : \mathbb{E}[\epsilon_i] = 0$
2. Дисперсия случайных ошибок одинакова и конечна, это свойство называется гомоскедастичностью:
$\forall i : \operatorname{Var}(\epsilon_i) = \sigma^2$
3. Случайные ошибки не связаны между собой:
$\forall i \neq j : \operatorname{Cov}(\epsilon_i, \epsilon_j) = 0.$

Линейная оценка $\widehat{\omega}$ весов w_i определяется следующим образом:

$$\hat{w}_i = \omega_{1i} y_1 + \omega_{2i} y_2 + \cdots + \omega_{ni} y_n$$

где каждое значение ω_{ki} зависит только от наблюдаемых данных X и почти наверняка является нелинейным. Так как оптимальными весами являются линейные оценки, то данная модель называется линейной регрессией. Дополнительно, оценка $\widehat{\omega}$ считается несмещенной, если математическое ожидание оценки равно истинному, но неизвестному значению оцениваемого параметра:

$$\mathbb{E}\left[\hat{w}_i\right] = w_i$$

Давайте посмотрим на, казалось бы, те же самые модели машинного обучения с другой стороны, проинтерпретировав их, как вероятностные.

Практически ни одна модель, которую мы создаем, не является безупречной. Однако можно объяснить это несовершенство по-разному.

Предположим, что мы решаем задачу регрессии, например, пытаемся предсказать годовую зарплату выпускника на основе его университетских оценок. Очевидно, что мы не сможем получить точную зависимость, по крайней мере, потому что мы не знаем многих факторов о выпускнике: где он работает, насколько он прилежен, каковы у него навыки soft skills и так далее. Как нам быть?

Первый вариант — можно признать, что идеальную модель найти не удастся, но стараться выучить наиболее оптимальную модель, которая максимально приблизит прогнозы к целевым значениям с учетом выбранной меры сходства, определенной на основе экспертных соображений. Таким образом, мы применяем инженерный подход к машинному обучению, где у нас есть формула с некоторыми параметрами, а также формализация понятия "приблизить" (функция потерь), и мы решаем задачу оптимизации по параметрам.

Второй вариант заключается в перекладывании ответственности за неточности наших предсказаний на случайность. Если мы не можем измерить что-то, то для нас это считается случайным фактором. В постановке задачи мы заменяем приближенное равенство на точное. Например, это может быть аддитивный шум, который чаще всего используется.

$$y = \langle x, w \rangle + \varepsilon$$

Здесь:

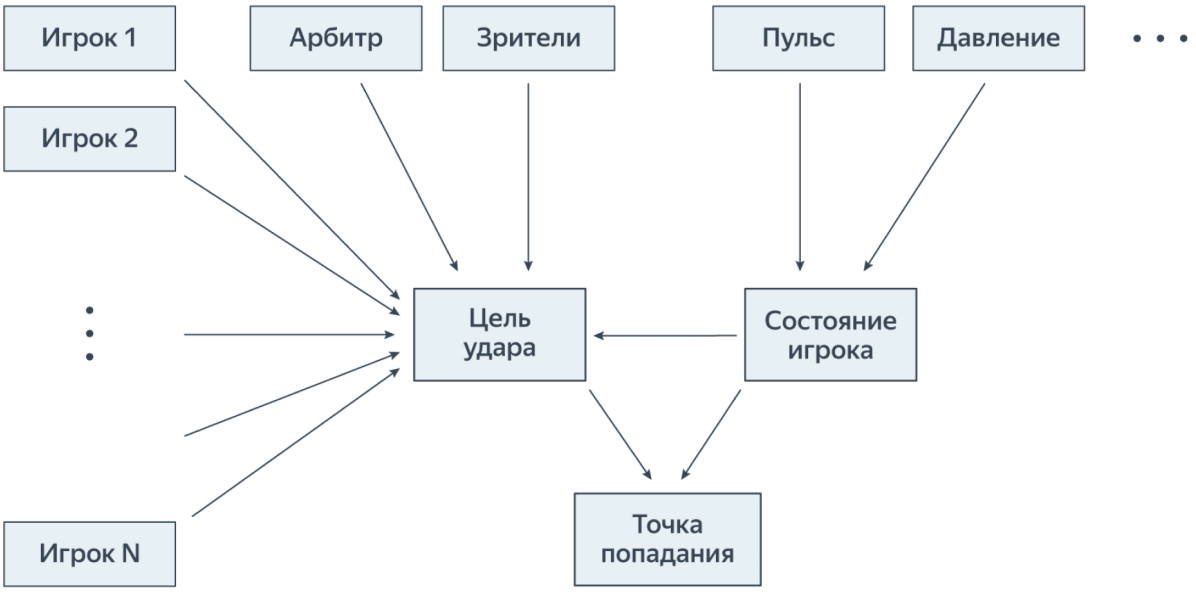
ϵ- некоторая случайная величина, которая представляет этот случайный шум. Таким образом, для каждого конкретного объекта x_i его истинное значение — это сумма $\langle x_i, w \rangle$

и конкретной реализации шума ϵ.

При построении такой модели мы можем выбирать различные распределения шума, чтобы указать, какая может быть ошибка. Чаще всего используется гауссовский шум с фиксированной дисперсией $\varepsilon \sim \mathcal{N}(0, \sigma^2)$, но могут быть и другие варианты.

На самом деле, для нашей задачи мы можем придумывать разные вероятностные модели $p(y|x, w)$, необязательно вида $y = fw(X) + \varepsilon$. Представьте, что мы хотим предсказывать точку на плоскости, в которую попадает мяч, бьющий по воротам футболист. Мы можем предположить, что эта точка имеет нормальное распределение со средним (целью удара), которое определяется ситуацией на поле и состоянием игрока, а также некоторой дисперсией (т.е. скалярной ковариационной матрицей), которая также зависит от состояния игрока и других сложных факторов, которые мы считаем случайными.

Состояние игрока — это сложное понятие, но мы, вероятно, можем его выразить, зная пульс, давление и другие физические показатели. В свою очередь, ситуацию на поле можно описать как функцию от позиций и движений других игроков, судей и зрителей — но все эти факторы невозможно перечислить, поэтому мы снова должны прибегнуть к случайности. Таким образом, мы получаем графическую модель.



Здесь стрелки указывают на статистические связи, а отсутствие стрелок означает, что статистическая независимость является предположением. Конечно, это всего лишь предположение, которое мы делаем для упрощения модели: ведь пульс человека и давление взаимосвязаны, так же как и поведение различных игроков на поле. Однако мы уже обсуждали, что каждая модель, включая вероятностную, является лишь приближенным отражением бесконечно сложного мира. Впрочем, если у нас есть достаточно вычислительных ресурсов, ничто не мешает нам попробовать учесть все пропущенные в настоящий момент связи.

Путем записи всего по определению условной вероятности мы получаем следующую вероятностную модель:

$$p\left(\begin{matrix}\text{точка}\\ \text{попадания}\end{matrix}\right) = \int p\left(\begin{matrix}\text{точка}\\ \text{попадания}\end{matrix}, \begin{matrix}\text{цель}\\ \text{удара}\end{matrix}, \begin{matrix}\text{состояние}\\ \text{игрока}\end{matrix}\right) d\left(\begin{matrix}\text{цель}\\ \text{удара}\end{matrix}\right) d\left(\begin{matrix}\text{состояние}\\ \text{игрока}\end{matrix}\right) =$$

$$= \int p\left(\begin{matrix}\text{точка}\\ \text{попадания}\end{matrix} \middle| \begin{matrix}\text{цель}\\ \text{удара}\end{matrix}, \begin{matrix}\text{состояние}\\ \text{игрока}\end{matrix}\right) p\left(\begin{matrix}\text{цель}\\ \text{удара}\end{matrix}, \begin{matrix}\text{состояние}\\ \text{игрока}\end{matrix}\right) d\left(\begin{matrix}\text{цель}\\ \text{удара}\end{matrix}\right) d\left(\begin{matrix}\text{состояние}\\ \text{игрока}\end{matrix}\right) =$$

$$= \int p\left(\begin{matrix}\text{точка}\\ \text{попадания}\end{matrix} \middle| \begin{matrix}\text{цель}\\ \text{удара}\end{matrix}, \begin{matrix}\text{состояние}\\ \text{игрока}\end{matrix}\right) p\left(\begin{matrix}\text{цель}\\ \text{удара}\end{matrix} \middle| \begin{matrix}\text{состояние ...}\\ \text{игрок 1, игрок 2}\\ \text{арбитр, зрители}\end{matrix}\right) p\left(\begin{matrix}\text{состояние}\\ \text{игрока}\end{matrix} \middle| \begin{matrix}\text{пульс}\\ \text{давление}\end{matrix}\right) \times$$

$$\times \left[\prod_i p(\text{игрок } i)\right] p(\text{арбитр})\, p(\text{зрители})\, p(\text{пульс})\, p(\text{давление}) \times$$

$$\times \left[\prod_i d(\text{игрок } i)\right] d(\text{арбитр})\, d(\text{зрители})\, d(\text{пульс})\, d(\text{давление})$$

Метод максимального правдоподобия

Метод максимального правдоподобия (Maximum Likelihood Estimation, MLE) это статистический метод оценки параметров модели, основанный на максимизации функции правдоподобия.

Основное предположение метода заключается в том, что наблюдаемые данные являются реализацией случайной выборки из распределения, параметры которого нужно оценить. Цель метода заключается в нахождении таких значений параметров, при которых вероятность получения наблюдаемого набора данных будет наибольшей.

Процесс оценки параметров с помощью метода максимального правдоподобия включает следующие шаги:
1. Формулирование статистической модели, которая описывает вероятностное распределение наблюдаемых данных в зависимости от неизвестных параметров.
2. Запись функции правдоподобия, которая показывает вероятность получения наблюдаемых данных при заданных значениях параметров.
3. Максимизация функции правдоподобия путем нахождения таких значений параметров, при которых функция достигает своего максимального значения.
4. Вычисление оценок параметров на основе найденных значений, и их интерпретация.

Преимущества метода максимального правдоподобия включают его простоту и эффективность. Оценки, полученные с помощью MLE (Maximum Likelihood Estimator

- позволяет нам определить оценку максимального правдоподобия), также обладают свойством состоятельности и асимптотической нормальности.

Однако важно учитывать, что метод максимального правдоподобия требует правильного выбора модели и предположений о распределении данных. В некоторых случаях, нарушение предположений может привести к недостоверным оценкам параметров.

Итак, мы стремимся найти значения параметров w, при которых модель $p(y|x,w)$ будет наиболее соответствовать обучающим данным. Метод максимального правдоподобия заключается в поиске такого значения, при котором вероятность (или плотность вероятности в случае непрерывной выборки) появления данной выборки будет максимальной.

$$\widehat{w}_{MLE} = \underset{w}{\operatorname{argmax}}\, p(y|X,w)$$

Величина $p(y|x,w)$ называется функцией правдоподобия (likelihood). Если мы считаем, что все объекты независимы, то функция правдоподобия распадается в произведение:

$$p(y|X,w) = p(y_1|x_1,w) \cdot \ldots \cdot p(y_i|x_i,w)$$

Теперь мы применяем логарифм функции правдоподобия, так как умножение сложно, а сложение легко. Кроме того, мы надеемся, что наши объекты, наблюдаемые в природе, имеют достоверность, отличную от нуля.

$$l(y|X,w) = \log p(y_1|x_1,w) + \ldots + \log p(y_i|x_i,w)$$

эту функцию мы так или иначе максимизируем по w, находя оценку максимального правдоподобия $\hat{w}$.

Как мы уже обсуждали $p(y_i|x_i,w) = p_\varepsilon(y - f_w(x_i))$. Следовательно,

$$l(y|X,w) = \sum_{i=1}^{N} \log p_\varepsilon(y_i - f_w(x_i))$$

Максимизация функции правдоподобия соответствует минимизации

$$\sum_{i=1}^{N} \left[-\log p_{\varepsilon}(y_i - f_w(x_i)) \right]$$

Это выражение можно трактовать как функцию потерь. Таким образом, можно сказать, что подбор параметров модели вероятностей с использованием метода максимального правдоподобия является эквивалентом "инженерной" оптимизации функции потерь.

Два важных аспекта: сходимость и эффективность.

Сходимость. При увеличении числа обучающих выборок к бесконечности, оценка максимального правдоподобия стремится к истинному значению параметра.
Эффективность. Оценка того, насколько близки мы к истинному параметру, осуществляется через ожидаемую среднеквадратичную ошибку, которая вычисляется как квадратичная разность между оценками и истинными значениями параметров. Для этого вычисляется математическое ожидание по m обучающим выборкам из данных, которые генерируют распределение.

Именно благодаря сходимости и эффективности оценка максимального правдоподобия часто считается предпочтительным методом для машинного обучения.

Разложение ошибки на смещение и разброс (Bias-variance decomposition)
Давайте дадим немного объяснений стандартных для машинного обучения понятий: смещение, разброс, переобучение и недообучение.

На следующей лекции мы рассмотрим несколько фундаментальных концепций машинного обучения. Во-первых, переобучение (overfitting) - это явление, при котором ошибка на тестовой выборке значительно превышает ошибку на обучающей выборке. Это основная проблема машинного обучения: если бы такого эффекта не было (ошибка на тесте примерно совпадала с ошибкой на обучении), то обучение сводилось бы к минимизации ошибки на тесте (так называемый эмпирический риск).

Во-вторых, недообучение (underfitting) - это явление, при котором ошибка на обучающей выборке достаточно велика, часто говорят "не удалось адаптироваться к выборке". Такое странное определение объясняется тем, что недообучение может наблюдаться при настройке алгоритмов итерационными методами (например,

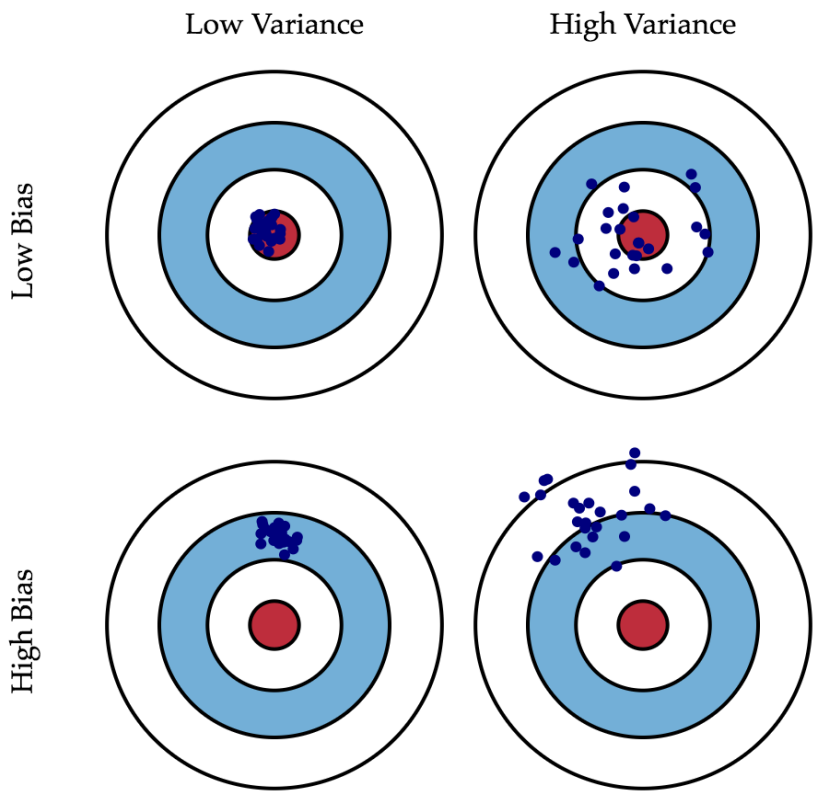
методом обратного распространения в нейронных сетях), когда сделано слишком мало итераций, то есть "не успели обучиться".

Дисперсию ответов алгоритмов мы называем разбросом, а матожидание разности между истинным ответом и выданным алгоритмом — смещением (bias). Ошибка разбивается на три компоненты. Первая связана с шумом в данных, в то время как две остальные связаны с используемыми моделью алгоритмами.

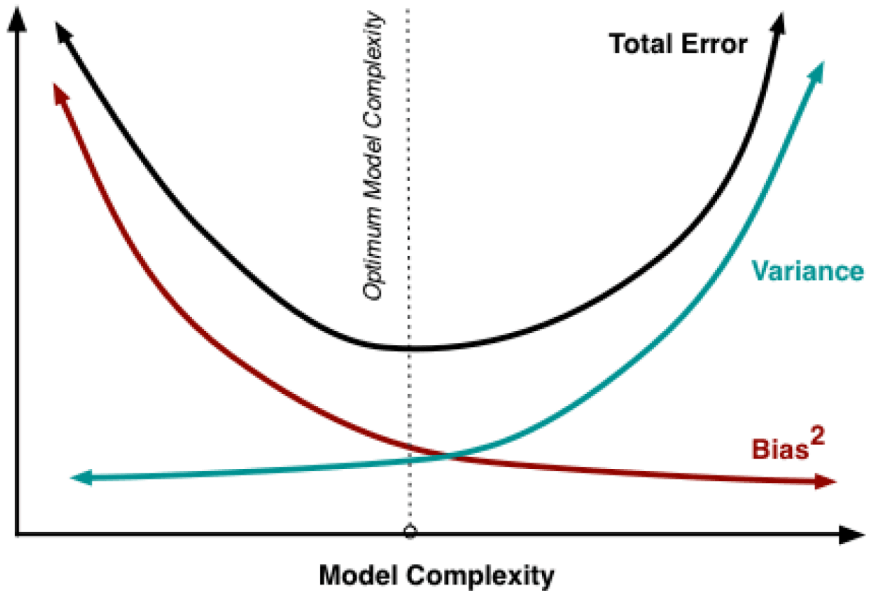
Очевидно, что разброс характеризует разнообразие алгоритмов (из-за случайности обучающей выборки, включая шум, и стохастической природы настройки), а смещение — способность модели алгоритмов адаптироваться к целевой зависимости.

Часто смещение и разброс можно проиллюстрировать следующим образом. Если провести аналогию, что алгоритм — это игрок в дартс, то самый лучший игрок будет иметь небольшое смещение и разброс — его дротики ложатся кучно в цель, если у игрока большое смещение, то они сгруппированы около другой точки, а если большой разброс, то они ложатся совсем не кучно. Эта аналогия понятна, но может вызывать путаницу. Например, известно, что спортсменов учат стрелять точно, поскольку нужного смещения легко достичь (например, целясь повыше). С алгоритмами машинного обучения все сложнее. Смещение нельзя просто "подвинуть", приходится переходить к другой (например, более сложной модели), а у нее может быть уже другой разброс.

Еще один важный момент: спортсмен может изменять точность своей цели, прицеливаясь выше/ниже/правее/левее, в то время как для алгоритма такие понятия не существуют. Напомним, что мы вводили понятия разброса и смещения в определенной точке. Если изменить смещение в этой точке, то модель будет вести себя по-другому и в других точках. Однако, если усреднить смещение, точнее его квадрат, по всем точкам, то мы получим просто число, которое не указывает, как изменить модель, чтобы уменьшить ошибку.



Теперь давайте рассмотрим наиболее распространенное иллюстрацию, которую используют для объяснения разброса и смещения. При увеличении сложности модели (например, степени полинома) ошибка на независимом контроле сначала уменьшается, а затем начинает увеличиваться. Обычно это объясняется уменьшением смещения (в сложных моделях есть много алгоритмов, которые хорошо описывают целевую зависимость) и увеличением разброса (в сложных моделях больше алгоритмов, и, следовательно, больше разброс).



Функции потерь в обучающих моделях определяются как разница между фактическим выходным значением y и прогнозируемым выходным значением ŷ. Эти функции также называются функциями ошибки или затрат.

Мы ранее обсуждали оценку параметров с помощью MLE, так как он предоставляет механизм для определения функции потерь.

Проще говоря, нам нужно научиться измерять точность модели и уменьшать ее ошибку, изменяя параметры модели. В данном случае параметры модели — это веса. Функция, которая оценивает, насколько часто модель ошибается, называется функцией потерь или функционалом качества. Важно, чтобы функцию потерь было легко оптимизировать: гладкая функция потерь хороша, а кусочно-постоянная функция потерь — плоха.

Функции потерь задачи регрессии

Существуют разные функции потерь. Выбор функции потерь определяет, насколько легко будет решать задачу и насколько близким будет предсказание модели к целевым значениям. Интуитивно понятно, что для нашей текущей задачи нужно взять векторы и предсказаний модели и сравнить, насколько они похожи

.

Поскольку эти векторы существуют в одном векторном пространстве, расстояние между ними может быть функцией потерь. Кроме того, положительная непрерывная функция от этого расстояния также может использоваться в качестве функции потерь. Количество способов определения расстояния между векторами также велико. Все эти варианты могут быть запутывающими, но мы обязательно обсудим это позже. Сейчас давайте возьмем квадрат L^2-нормы разницы между предсказаниями модели и у истинными значениями вектора в качестве функции потерь.

Во-первых, как мы увидим далее, решить эту задачу будет легко, а во-вторых, у этой функции потерь также есть несколько дополнительных свойств:

- L^2-норма разницы – это евклидово расстояние $|y - f(x)|^2$ между вектором таргетов и вектором ответов модели, то есть мы их приближаем в смысле самого простого и понятного «расстояния».

- С точки зрения статистики это соответствует гипотезе, что наши данные состоят из линейного «сигнала» и нормально распределенного «шума». (Сигнал обычно относится к входным данным, предоставляемым модели для обучения или прогнозирования. Сигнал представляет собой числовую или категориальную информацию, которая содержится в данных и помогает модели понять и сделать предсказания. Входные сигналы могут быть представлены различными форматами, такими как числовые значения, изображения, аудиофайлы или текстовые данные. Модели машинного обучения используют сигналы, чтобы обнаружить закономерности и шаблоны в данных, чтобы принимать решения и делать прогнозы. Нормально распределенный шум - это тип случайного сигнала или данных, который следует нормальному (гауссову) распределению вероятностей. Такой шум имеет симметричную форму и характеризуется средним значением (математическим ожиданием) и стандартным отклонением. В нормально распределенном шуме значения, наиболее вероятные, находятся вокруг среднего значения, а чем дальше от среднего значения, тем меньше вероятность получить такое значение.)

Так вот, наша функция потерь выглядит так:

$$L(f, X, y) = |y - f(X)|_2^2 = \|y - Xw\|_2^2 = \sum_{i=1}^{N} (y_i - \langle x_i, w \rangle)^2$$

Такой функционал ошибки не очень хорош для сравнения поведения моделей на выборках разного размера. Гораздо лучше посмотреть на среднеквадратичное отклонение

$$L(f, X, y) = \frac{1}{N} \sum_{i=1}^{N} (y_i - \langle x_i, w \rangle)^2$$

Данная функция потерь называется Mean Squared Error, MSE или среднеквадратическим отклонением. Разница с L^2 -нормой чисто косметическая, на алгоритм решения задачи она не влияет:

$$MSE(f, X, y) = \frac{1}{N} |y - Xw|_2^2$$

В самом широком смысле, функции работают с объектами множеств: они берут некий входной объект из одного множества и выдают на выходе соответствующий ему объект из другого множества. Если мы имеем дело с отображением, которое на вход принимает функции и на выходе выдаёт число, то такое отображение называется функционалом. Если вы посмотрите на нашу функцию потерь, то увидите, что это именно функционал. Для каждой конкретной линейной функции, которую задают веса ω_i, мы получаем число, которое оценивает, насколько точно эта функция приближает наши значения y. Чем меньше это число, тем точнее наше решение. Значит, чтобы найти лучшую модель, нам нужно минимизировать этот функционал по весам ω.

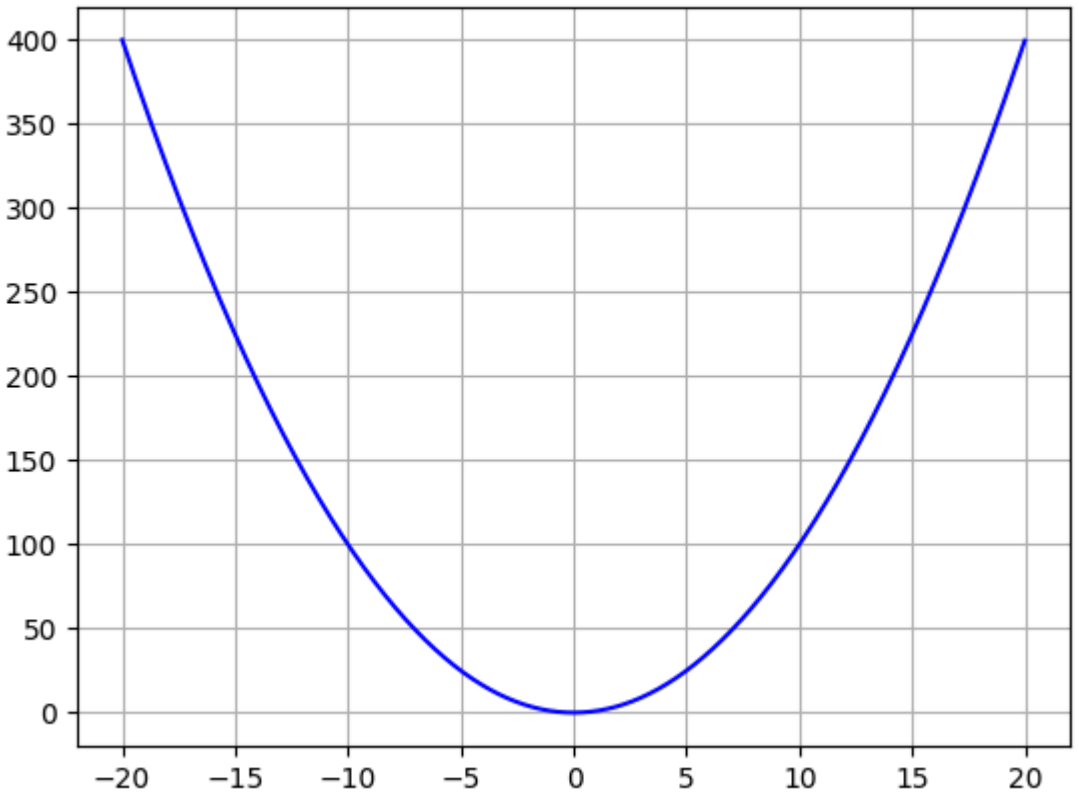
Функция потерь и метрика качества имеют важное различие.

- Суть функции потерь заключается в том, что мы сводим задачу построения модели к задаче оптимизации. Обычно требуется, чтобы функция потерь обладала хорошими свойствами, такими как дифференцируемость.

- Метрика качества, с другой стороны, является внешним и объективным критерием, который зависит не от параметров модели, а только от предсказанных меток.

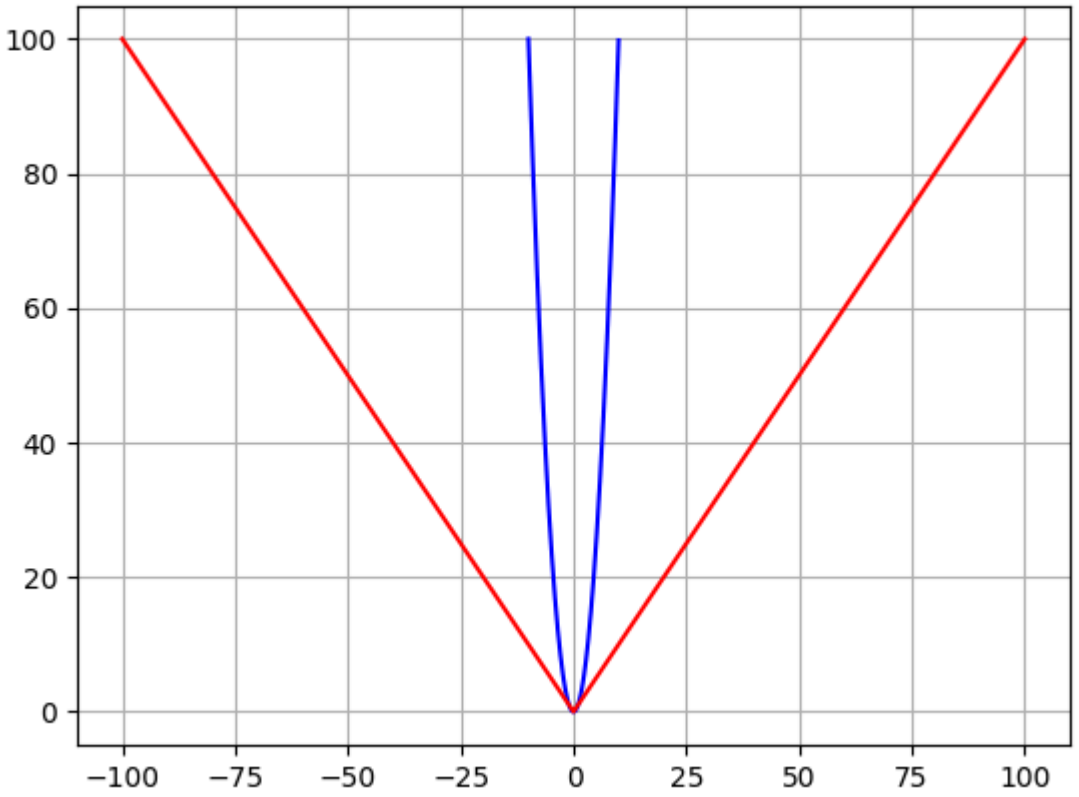
В некоторых случаях метрика может совпадать с функцией потерь. Например, в задаче регрессии MSE играет роль как функции потерь, так и метрики.

Также Вы можете встретить функцию потерь MSE как $L(y, f) = (y - f)^2$, оно же L_2-loss, оно же Gaussian loss. Это классическое условное среднее, самый частый и простой вариант. Если нет никакой дополнительной информации или требований к устойчивости (робастности) модели — используйте его



Еще одна функция потерь:
Средняя абсолютная ошибка (MAE) представляет собой среднее значение абсолютных отклонений между предсказаниями модели и фактическими значениями данных. Она вычисляется путем нахождения разницы между предсказаниями и истинными значениями, применения абсолютного значения к этой разнице и усреднения полученных значений по всему набору данных.

В отличие от MSE, MAE не имеет квадратичной зависимости от ошибок и, следовательно, не уделяет большое внимание крупным ошибкам. Это означает, что MAE более устойчива к выбросам и чувствительна к малым ошибкам.



MAE (красный) и MSE (синий) функции потерь

$L(y, f) = |y - f|$, оно же $L - 1$loss. Эта, на первый взгляд, не очень дифференцируемая вещь, на самом деле определяет условную медиану. Медиана, как мы знаем, более устойчива к выбросам, поэтому в некоторых задачах эта функция потерь предпочтительнее, так как она не так сильно штрафует большие отклонения, нежели квадратичная функция.

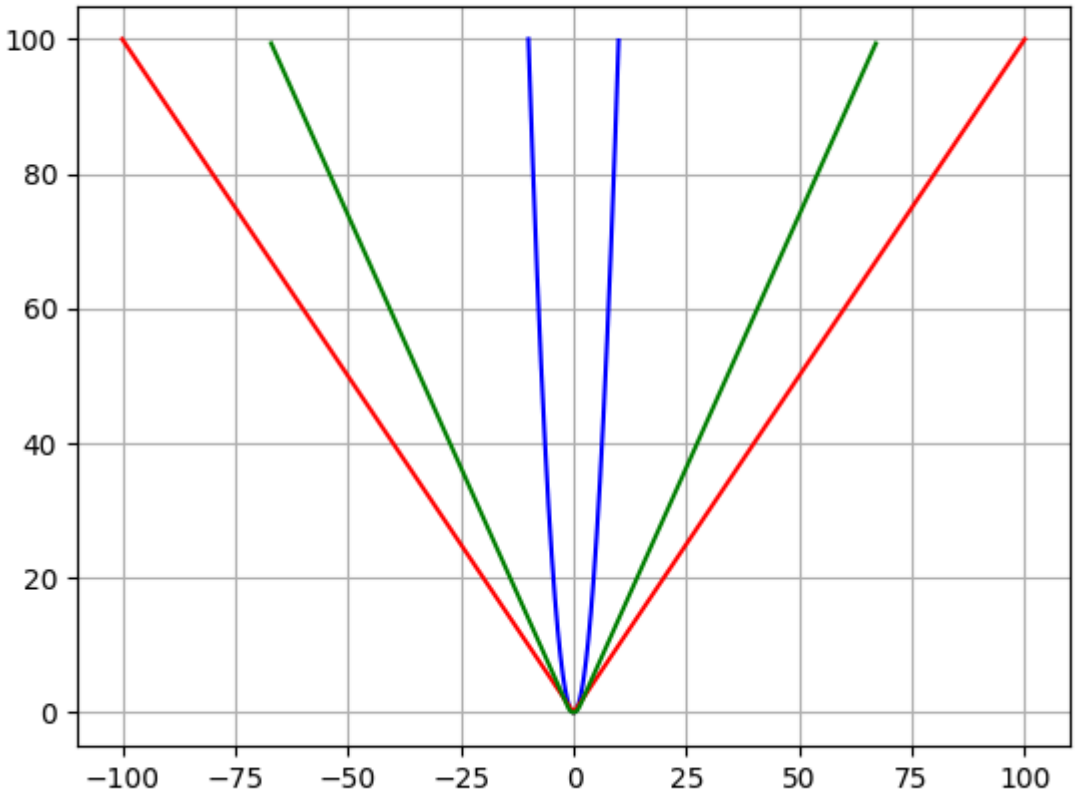
Теперь мы знаем, что MSE отлично подходит для изучения выбросов, а MAE - для их игнорирования. Но как насчет чего-то посередине?

Функция Huber Loss предлагает компромиссное решение между MSE и MAE. Мы можем определить ее с помощью следующего кусочного уравнения:

В основном, это уравнение гласит: для значений потерь, меньших чем дельта, используйте MSE; для значений потерь, больших чем дельта, используйте MAE. Таким образом, эта функция эффективно объединяет лучшие стороны обоих функций потерь!

Использование MAE для больших значений потерь уменьшает влияние выбросов, поэтому мы все равно получаем модель с хорошим округлением. В то же время, мы используем MSE для меньших значений потерь, чтобы сохранить квадратичную функцию близкой к центру.

Это приводит к увеличению значений потерь до тех пор, пока они не превысят 1. Как только потеря для этих данных падает ниже 1, квадратичная функция уменьшает их влияние, чтобы сосредоточить обучение на данных с более высокой ошибкой.



Функции потери MAE (красный), MSE (синий) и Хьюбер (зеленый)

Функции потерь задачи классификации

Можно вспомнить, как возникла среднеквадратическая ошибка (MSE) в статистике. При обучении классической линейной регрессии оптимизация MSE эквивалентна нахождению ответа методом максимального правдоподобия. Что если мы пойдем таким же путем для задачи классификации на два класса? Мы получим бинарную кросс-энтропию (BCE - Binary Cross-Entropy)! В этом смысле MSE и BCE - результаты одного и того же метода, примененного к разным задачам. С помощью бинарной кросс-энтропии (также известной как logloss) можно решать и задачу мультилейбл классификации — достаточно для каждого класса иметь отдельную голову, определяющую, относится ли объект к данному классу.

Тогда конечная функция потерь будет суммой бинарных кросс-энтропий для каждого класса. В общем, это очень универсальный инструмент с математическим обоснованием, неудивительно, что его используют чаще других функций.

Задача функции потерь в классификации заключается в определении стоимости неточности предсказаний классификационной модели. Для выполнения бинарной классификации с использованием логистической регрессии, функция потерь может быть рассчитана следующим образом:

$$LogLoss = -\frac{1}{l}\sum_{i=1}^{l}(y_i \cdot log(\widehat{y_i}) + (1 - y_i) \cdot log(1 - \widehat{y_i}))$$

где l - размер выборки, $y_i = \{0,\ 1\}$ - бинарная метка класса, заданная в примере, y.
Функция потерь получается путем суммирования логарифма потерь на каждом примере, что можно легко заметить. Потери на каждом примере определяются так: если предсказанный класс совпадает с фактическим, то потери равны 0, в противном случае потери равны 1. Ясно, что чем больше неправильных классификаций, тем выше значение LogLoss и тем хуже модель. Поэтому, чтобы получить лучшую модель, необходимо минимизировать функцию потерь.

Преимущество метрики LogLoss заключается в ее устойчивости к выбросам и аномальным значениям в данных, а также в ее простоте вычисления. Недостатком является сложность интерпретации из-за нелинейного характера.

Cross-Entropy

Однако, если у нас стоит задача мультиклассовой классификации, где количество классов больше двух и они являются взаимоисключающими, то невозможно научиться классифицировать каждый класс отдельно и выбирать класс с максимальным вероятностью, так как уверенность в нейронных сетях сама по себе не калибруется. Невозможно также использовать argmax для нахождения максимума среди вероятностей и штрафовать за неверный индекс, так как максимум не является дифференцируемым.

Однако ничто не мешает нам обобщить бинарную кросс-энтропию до категориальной, которая также называется просто кросс-энтропией. Для этого мы заменим несколько сигмоид (по одной для каждого класса) на softmax (Функция softmax используется для преобразования чисел в вероятности. Она принимает на вход набор чисел и возвращает новый набор чисел, каждое из которых представляет собой вероятность относительно других чисел. Функция делит каждое число на сумму всех чисел, чтобы получить долю, или вероятность, которую оно составляет относительно остальных чисел. Это полезно, когда мы хотим получить распределение вероятностей для набора значений).

Уверенность всё ещё будет находиться в диапазоне от 0 до 1, но сумма всех выходов будет равна единице. Теперь мы можем вычислять потери только для положительного класса, так как остальные выходы участвуют в функции софтмакс и через них проходит поток градиента. После обучения, при использовании модели, мы можем просто выбирать максимальное значение из всех вероятностей.

$$CE(p_t) = -log(p_t)$$

По сути, функция бинарной перекрестной энтропии (BCE) также вычисляется только для "позитивного" класса. Если целевое значение yt = 1, то "позитивным" считается класс "единица" и его вероятность равна pt. Если целевое значение yt = 0, то "позитивным" считается класс "ноль", и его вероятность равна 1 - pt. Поэтому функции перекрестной энтропии (CE) и бинарной перекрестной энтропии (BCE) часто путают между собой :)

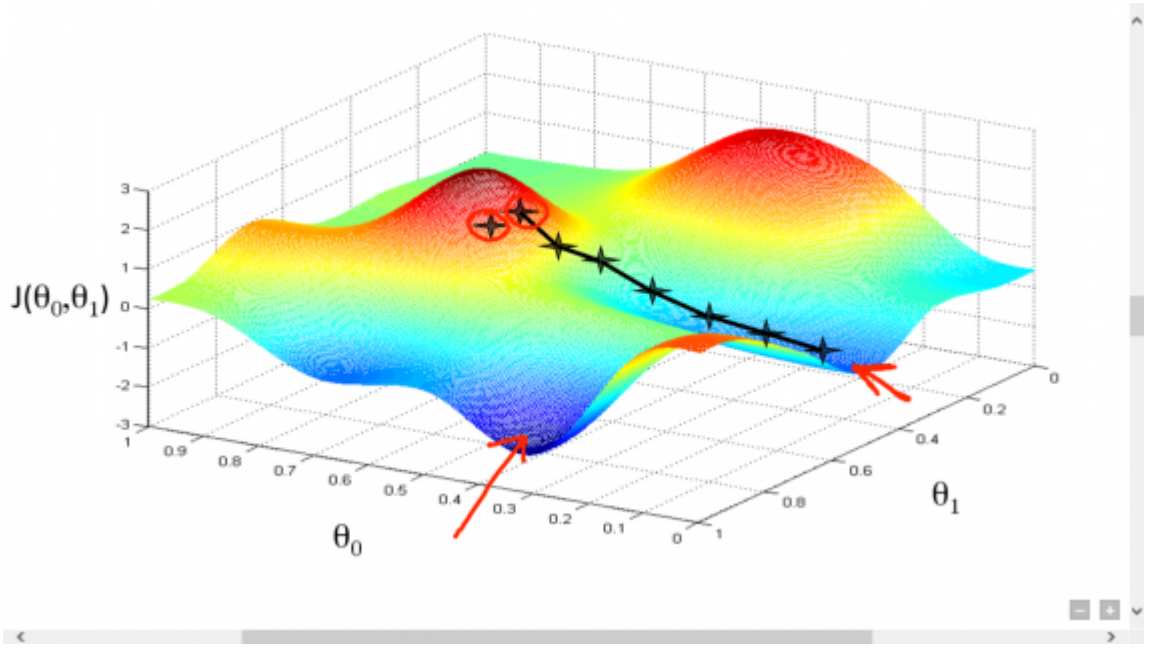
Градиентный спуск и методы оптимизации

Итак, как все-таки обучается модель? С помощью градиентного спуска:

Градиентный спуск является наиболее распространенным алгоритмом обучения нейронных сетей и применяется практически во всех моделях машинного обучения. Метод градиентного спуска с некоторыми изменениями широко используется для обучения персептрона и глубоких нейронных сетей и известен как метод обратного распространения ошибки.

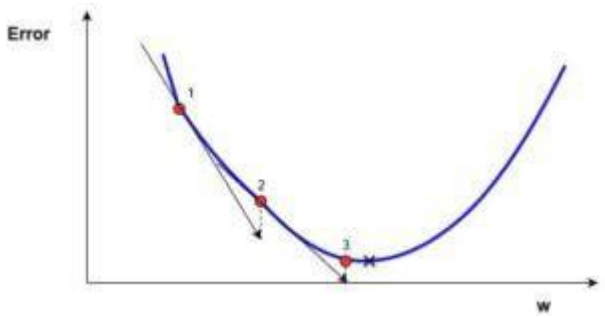
Градиентный спуск — это метод поиска минимального значения функции потерь. Этот метод используется для минимизации любой функции, чтобы найти самую глубокую впадину в этой функции. Функция потерь используется для контроля ошибки в прогнозах модели машинного обучения.

Поиск минимума означает получение наименьшей возможной ошибки или повышение точности модели. Для этого мы перебираем набор учебных данных и настраиваем параметры модели (веса и смещения), чтобы улучшить точность. Таким образом, градиентный спуск помогает нам минимизировать функцию потерь. Этот алгоритм можно представить как спуск во впадину в попытке найти "золото" на дне ущелья, то есть самое низкое значение ошибки.



Для того чтобы найти самую низкую ошибку (глубочайшую впадину) в функции потерь (относительно одного веса), необходимо настроить параметры модели. Как это делается? Для этого применяется математический анализ. Анализ позволяет определить, что наклон графика функции является производной функции по переменной. Этот наклон всегда указывает на ближайшую впадину!

На рисунке представлен график функции потерь (обозначенный как "Ошибка" с символом "J") относительно одного веса. Если мы вычислим наклон (обозначим это как dJ/dw) функции потерь по одному весу, то получим направление, в котором необходимо двигаться, чтобы достичь локальных минимумов. Давайте пока представим, что наша модель имеет только один вес.



💡 Важно: при переборе всех учебных данных мы продолжаем добавлять значения dJ/dw для каждого веса. Поскольку потери зависят от примера обучения, dJ/dw также продолжает изменяться. Затем мы делим собранные значения на количество примеров обучения, чтобы получить среднее значение. Затем мы используем это среднее значение для настройки каждого веса.

🔥 Также обратите внимание: функция потерь предназначена для отслеживания ошибки с каждым примером обучения, в то время как производная функции относительно каждого веса показывает, насколько нужно изменить вес, чтобы минимизировать ошибку для данного примера обучения. Вы можете создавать модели даже без использования функции потерь, но вам все равно потребуется производная относительно каждого веса (dJ/dw).

Теперь, после определения направления, в котором необходимо подтолкнуть вес, необходимо разобраться, как это сделать. Для этого мы используем гипер-параметр — коэффициент скорости обучения. Гипер-параметр представляет собой значение, необходимое вашей модели, о котором у нас есть очень смутное представление. Обычно эти значения можно изучить методом проб и ошибок.

Однако здесь нет универсального подхода для всех гипер-параметров. Коэффициент скорости обучения можно рассматривать как "шаг в правильном направлении", где направление определяется посредством dJ/dw.

Это была функция потерь, построенная для одного веса. В реальной модели мы выполняем все перечисленные выше шаги для всех весов, перебирая все примеры обучения. Даже в относительно небольшой модели машинного обучения у вас будет больше чем 1 или 2 веса. Это затрудняет визуализацию, так как график будет иметь размеры, которые разум не может представить.

О градиенте

Помимо функции потерь, для метода градиентного спуска необходим градиент dJ/dw (производная функции потерь по одному весу, выполняемая для всех весов). dJ/dw зависит от выбора функции потерь. Наиболее распространенной функцией потерь является функция потерь среднеквадратичной ошибки.

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

Производная этой функции относительно любого веса (эта формула показывает вычисление градиента для линейной регрессии):

$$\frac{\partial}{\partial\theta_j}\,\mathrm{MSE}\,(\theta) = \frac{2}{m}\sum_{i=1}^{m}(\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)})\,x_j^{(i)}$$

Вся математика в ГС заключается в следующем: ГС на самом деле не содержит большого количества математики. Единственные математические операции, которые включены в ГС — это умножение и деление, к которым мы доберемся. Таким образом, ваш выбор функции будет влиять на вычисление градиента каждого веса.

Коэффициент скорости обучения

Формулы градиентов для каждой функции потерь можно найти в интернете, не имея навыков их вывода самостоятельно.

Однако у большинства моделей возникают проблемы с коэффициентом скорости обучения.

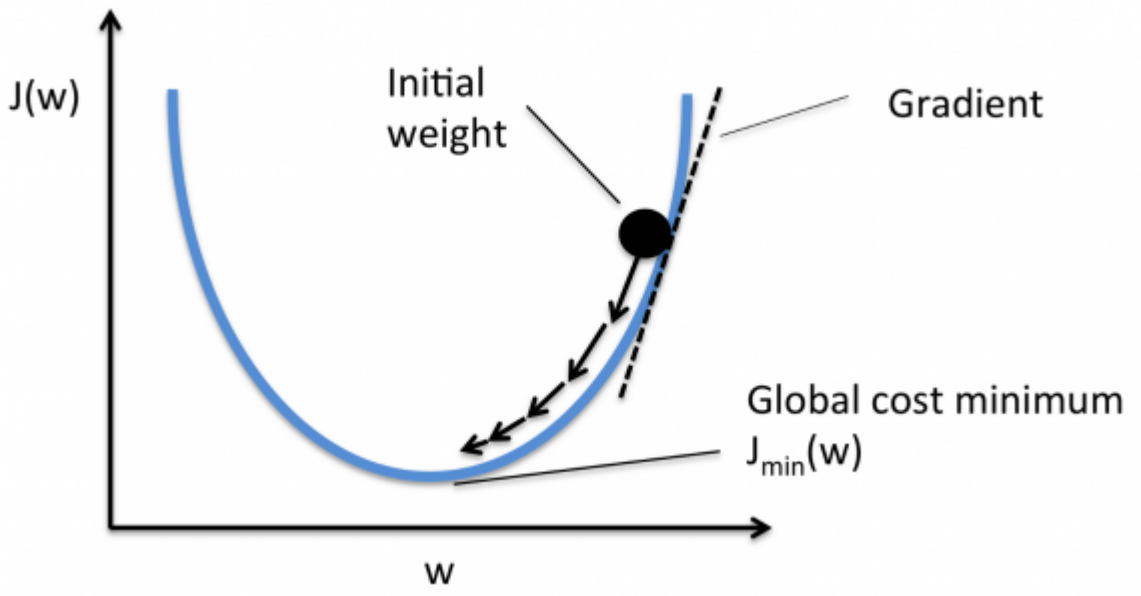
Коэффициент скорости обучения — это гиперпараметр, определяющий порядок корректировки весов с учетом функции потерь градиентного спуска. Чем меньше значение, тем медленнее градиентный спуск. Использование низкого коэффициента скорости обучения дает положительный эффект в том смысле, что локальные минимумы не пропускаются, но при этом приходится тратить больше времени на сходимость, особенно при достижении области плато.
Давайте рассмотрим обновленное выражение для каждого веса (j находится в диапазоне от 0 до количества весов, а Theta-j - это j-й вес в векторе весов, k находится в диапазоне от 0 до количества смещений, где B-k - это k-е смещение в векторе смещений). Здесь alpha — коэффициент скорости обучения. Мы вычисляем dJ/dTheta-j (градиент веса Theta-j), а затем делаем шаг размера alpha в этом направлении. Таким образом, мы движемся вниз по градиенту. Чтобы обновить смещение, замените Theta-j на B-k.

$$\text{Repeat until convergence } \{$$

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

$$\}$$

Если величина шага alpha слишком велика, то минимум преодолевается, т.е. минимум пропускается; если alpha слишком мала, то для достижения минимума требуется слишком много итераций. Поэтому величина alpha должна быть соответствующей.



Давайте рассмотрим использование градиентного спуска более подробно. При использовании градиентного спуска мы проходимся по каждому примеру обучения и вычисляем градиент функции потерь по каждому весу и смещению. Это позволяет нам получить вектор, содержащий градиенты для каждого веса и переменную, содержащую градиент смещения.

Затем мы добавляем эти градиенты в аккумулятивный вектор весов. После каждой итерации по всем примерам обучения, этот вектор будет содержать сумму градиентов каждого веса за несколько итераций. Аналогично, мы также добавляем градиент смещения к аккумулятивной переменной.

Таким образом, использование градиентного спуска позволяет нам эффективно оптимизировать веса и смещение модели, обновляя их по мере прохождения через каждый пример обучения.

После того как вы перебрали все примеры обучения, выполните следующие шаги:

1. Разделите аккумулятивные переменные весов и смещений на количество примеров обучения. Таким образом, вы получите средние градиенты для всех весов и средний градиент для смещения. Назовем их обновленными аккумуляторами (OA).

2. Затем, используя следующую формулу, обновите все веса и смещение. Вместо dJ / dTheta-j подставьте обновленные аккумуляторы для весов и смещения. То же самое проделайте для смещения.

Это только одна итерация градиентного спуска.

Повторите этот процесс снова и снова для заданного количества итераций. Это означает, что для первой итерации градиентного спуска вы перебираете все примеры обучения, вычисляете градиенты, затем обновляете веса и смещения. Затем повторяете это для заданного количества итераций градиентного спуска.

Модификации градиентного спуска

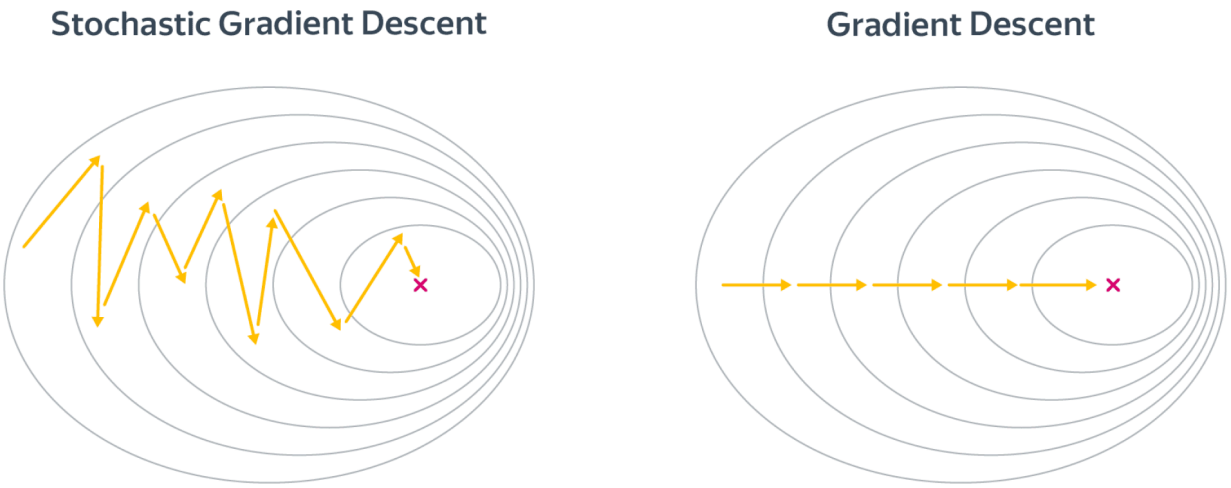
Существует несколько вариантов градиентного спуска

Стохастический градиентный спуск: вместо того, чтобы перебирать и использовать весь набор примеров обучения, мы применяем подход "используй только один". Термин "стохастический" означает случайность, на которой основан алгоритм. В SGD вместо всего набора данных для каждой итерации, мы случайным образом выбираем пакеты – несколько образцов из набора.
Здесь есть несколько моментов, которые следует отметить:

1. Необходимо перемешать набор примеров обучения перед каждым проходом в ГС, чтобы каждый раз перебирать их в случайном порядке.
2. Поскольку каждый раз используется только один пример обучения, ваш путь к локальному минимуму будет неоптимальным.
3. С каждой итерацией ГС необходимо перемешивать набор обучения и выбирать случайный пример обучения.
4. Поскольку вы используете только один пример обучения, ваш путь к локальным минимумам будет шумным.

Еще одно преимущество этого метода — возможность работы с внешней памятью. Это связано с тем, что образец может быть настолько большим, что поместится только на жестком диске. В таких случаях стоит выбирать достаточно большой размер выборки. Доступ к данным с диска всегда медленнее, чем к данным из оперативной памяти, поэтому лучше получить сразу больше данных.

Поскольку стохастический градиент является лишь оценкой истинного градиента, SGD может быть достаточно шумным:

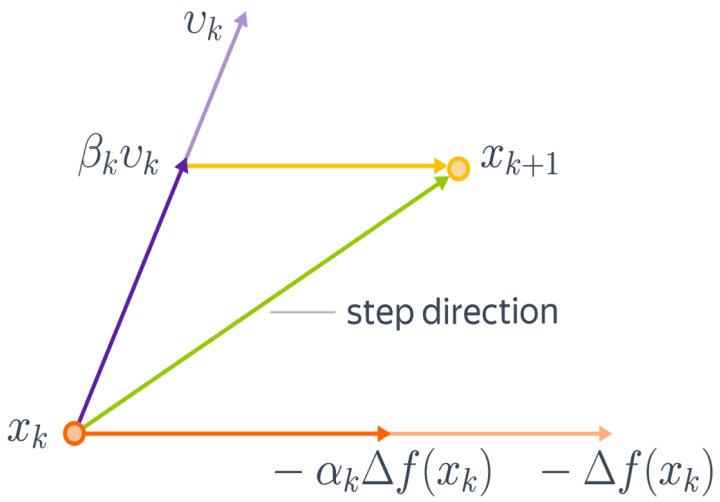


Использование информации о предыдущих шагах

Еще одна претензия к методу градиентного спуска заключается в том, что мы не используем информацию о предыдущих шагах, хотя, возможно, она может содержать что-то полезное.

Метод инерции, momentum

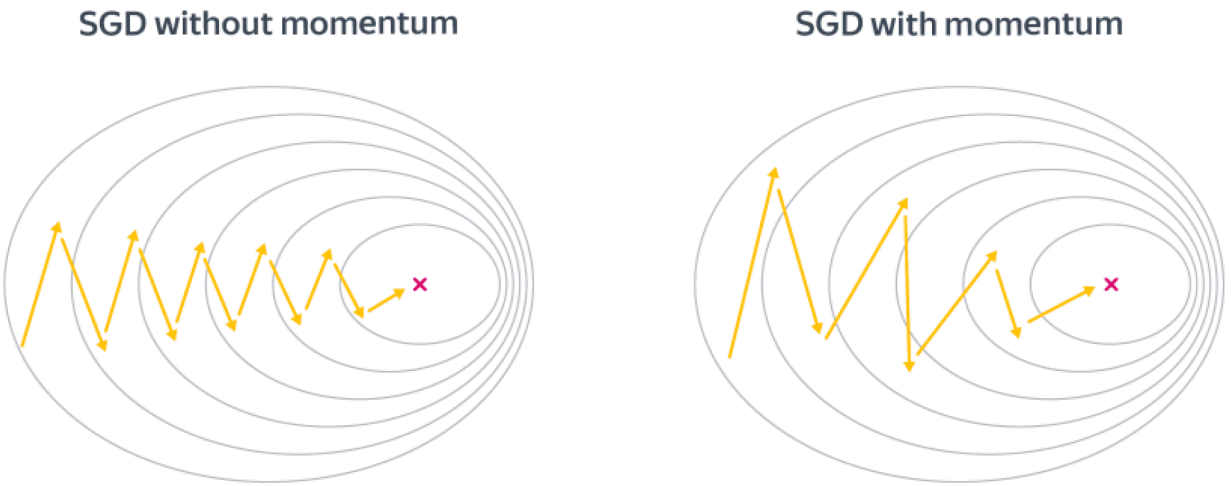
Давайте начнем с физической аналогии. Представьте себе мячик, который катится с горы. В данном случае гора — это график функции потерь в пространстве параметров нашей модели, а мячик — ее текущее значение. Реальный мячик не остановится перед небольшим уступом, поскольку у него есть некоторая масса и уже накопленный импульс — он может двигаться даже вверх по склону в течение некоторого времени. Аналогичный подход может быть использован и в градиентной оптимизации. В англоязычной литературе этот метод называется Momentum.



Momentum

С математической точки зрения, мы добавляем к градиентному шагу еще одно слагаемое:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k) + \beta_k (x_k - x_{k-1}).$$



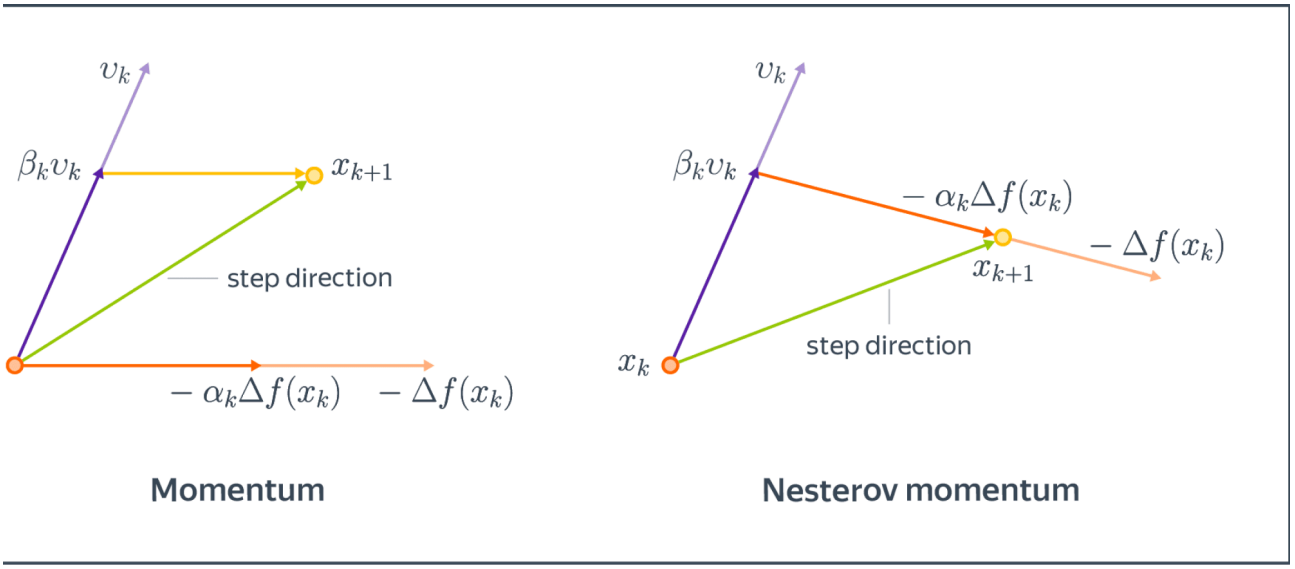
Accelerated Gradient Descent (Nesterov Momentum)

Рассмотрим некоторую дополнительную модификацию, которая была предложена в качестве оптимального метода первого порядка для решения выпуклых оптимизационных задач.

В 1983 году Ю. Нестеров предложил алгоритм с порядково-оптимальным оценщиком. Для этого несколько модифицируем импульс и будем рассматривать градиент не как текущую точку, а как точку, движущуюся относительно импульса:

$$v_{k+1} = \beta_k v_k - \alpha_k \nabla f(x_k + \beta_k v_k) x_{k+1} = x_k + v_{k+1}$$

Сравним с обычным momentum:



Комментарий: иногда говорят, что Nesterov Momentum "предсказывает будущее" и исправляет ошибки на текущем шаге оптимизации. Естественно, никто буквально не предсказывает будущее.

В работе Нестерова были предложены определенные (и довольно волшебные) значения для импульса, которые получаются из еще более волшебной последовательности. Мы не будем их приводить, так как нас в первую очередь интересует невыпуклый случай.

Nesterov Momentum позволяет значительно улучшить устойчивость и скорость сходимости в некоторых случаях. Однако, конечно, он не является универсальным методом в задачах оптимизации, хотя в выпуклом мире он считается теоретически непревзойденным.

Также стоит отметить, что ускоренный метод может быть применен непосредственно к проксимальному градиентному спуску.

Адаптивный подбор размера шага — это способ выбора оптимального значения для шага в алгоритме градиентного спуска. Вместо того чтобы использовать фиксированное значение шага, мы хотим адаптировать его в процессе обучения.

Адаптивный подбор размера шага особенно важен в случае стохастического градиентного спуска (SGD), так как вычисление значения функции потерь в каждой точке может быть очень дорого. Поэтому мы не можем просто использовать методы наискорейшего спуска.

Вместо этого, нам нужно использовать более умные методы. Один из таких методов — Adagrad. Он является адаптацией стохастического градиентного спуска.

Adagrad — это алгоритм, который адаптирует размер шага в зависимости от истории градиентов. Он учитывает предыдущие значения градиентов и делает шаги, которые более подходят для каждого параметра.

В результате Adagrad позволяет эффективно подбирать размер шага для каждого параметра, что может улучшить скорость и качество обучения.

Идея заключается в следующем: если мы достигли плато по определенной координате и соответствующая компонента градиента начала затухать, то нам не следует слишком сильно уменьшать размер шага, поскольку есть риск застрять на этом плато. Однако нам все же необходимо уменьшать размер шага, поскольку это плато может содержать оптимум. Если градиент в течение длительного времени остается достаточно большим, это может быть сигналом о необходимости уменьшить размер шага, чтобы не упустить оптимум. Поэтому мы стремимся компенсировать слишком большие или слишком маленькие значения компонент градиента.

Однако часто возникает ситуация, когда размер шага слишком быстро уменьшается. Для решения этой проблемы был разработан другой алгоритм. Adagrad. Данный модуль реализует принцип работы адаптивного градиентного спуска (adaptive gradient - AdaGrad).

Изменим предыдущую идею следующим образом: вместо простого сложения норм градиентов мы будем усреднять их в скользящем режиме.

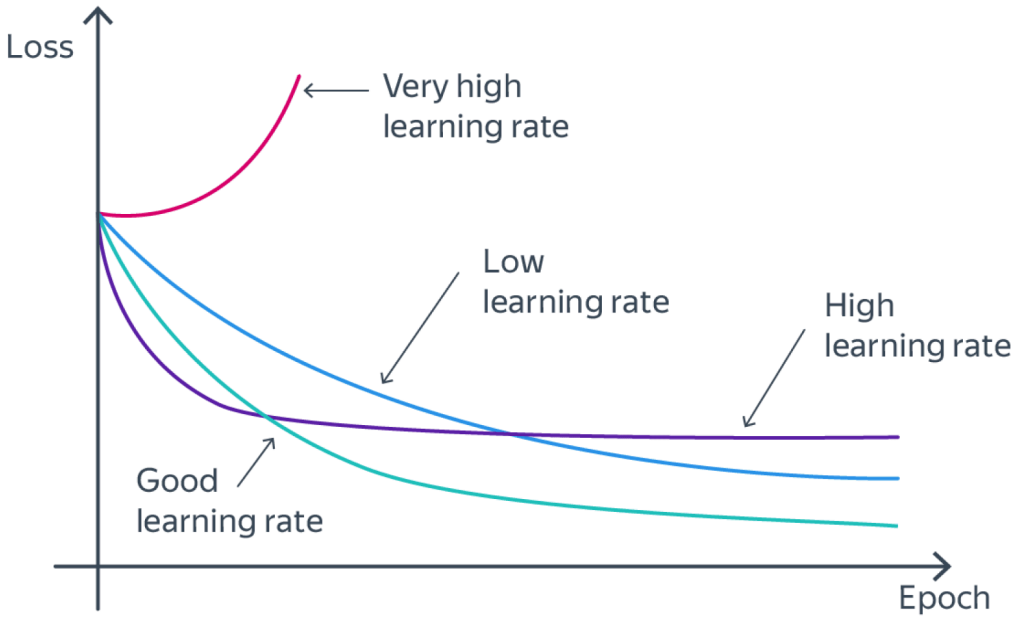
$$G_{k+1} = \gamma G_k + (1-\gamma)(\nabla f(x_k))^2$$

$$x_{k+1} = x_k - \frac{\alpha}{\sqrt{G_{k+1} + \varepsilon}} \nabla f(x_k).$$

Этот выбор позволяет сохранить историю градиентов, но при этом уменьшает скорость уменьшения размера шага.

Общий вывод:

Благодаря адаптивному выбору шага в современных оптимизаторах нет необходимости подбирать последовательность α_k размеров всех шагов, достаточно выбрать только одно число - learning rate α, и алгоритм самостоятельно будет делать все остальное. Однако выбор learning rate все еще требует осторожности: алгоритм может либо слишком рано достичь плато, либо полностью расходиться. Пример приведен на иллюстрации ниже.



Мы объединяем все вместе...

Адам

Теперь мы покажем ключевой момент нашей программы: алгоритм Адам, который считается стандартным решением и практически идеальным инструментом в задачах стохастической оптимизации.

Алгоритм градиентного спуска Adam (Adaptive Moment Estimation) является модификацией классического градиентного спуска и обеспечивает эффективный и быстрый процесс оптимизации функции.

Основная идея Adam заключается в комбинировании двух методов - адаптивного шага обучения и моментов.

1. Адаптивный шаг обучения:

Adam адаптивно регулирует скорость обучения для каждого параметра на основе истории градиентов. Это позволяет выбрать оптимальный шаг обучения для каждого параметра отдельно и избежать проблем со сходимостью на плоских участках функции потерь.

2. Моменты низкого порядка:

Adam использует два момента низкого порядка — первый момент (mean) и второй момент (variance) градиентов. Они используются для создания одномерной оценки среднего и дисперсии градиентов.

Алгоритм работы Adam следующий:

1. Инициализация переменных: задаются начальные значения для средних моментов градиентов (mean) и квадратов моментов (variance).

2. Подсчет градиентов: вычисляются градиенты по каждому параметру на основе функции потерь.

3. Обновление средних моментов: вычисляются новые значения средних моментов градиентов, учитывая текущие градиенты.

4. Обновление квадратов моментов: вычисляются новые значения квадратов моментов градиентов, учитывая текущие градиенты.

5. Коррекция смещения: исправляется смещение в оценке средних моментов путем деления этих оценок на (1 минус beta1) и (1 минус beta2) соответственно.

6. Обновление параметров: параметры модели обновляются в соответствии с вычисленными градиентами и адаптивным шагом обучения.

7. Повторение шагов 2-6 до достижения желаемого результата или определенного числа итераций.

Adam обычно имеет очень хорошую сходимость и может быть эффективен при обучении нейронных сетей и других сложных моделей. Однако иногда может потребоваться экспериментирование с параметрами алгоритма для достижения наилучшего результата для конкретной задачи.

Название Адам = ADAptive Momentum намекает на то, что мы объединяем идеи двух последних разделов в один алгоритм.

Также следует помнить, что для достижения более быстрой сходимости (с точки зрения количества итераций/объема рассмотренных данных) требуются большие объемы памяти, так как Adam требует хранения как параметров модели, так и градиентов, накопленного импульса и нормировочных констант (cache). Если вы решите продолжить обучение модели, остановленное на некоторой точке, необходимо восстановить из чекпоинта не только веса модели, но и накопленные параметры Adam. В противном случае оптимизатор начнет собирать все свои статистики с нуля, что может сильно повлиять на качество дообучения. То же самое

относится ко всем описанным выше методам, так как каждый из них накапливает определенные статистики во время обучения.

Выводы:

В данной лекции мы рассмотрели понятие функции потерь и оптимизации, которые являются важными инструментами в машинном обучении. Функция потерь представляет собой математическую функцию, которая измеряет разницу между предсказанными и фактическими значениями модели. Она позволяет оценить, насколько хорошо модель работает и какие параметры нужно изменить для улучшения ее результатов.

Оптимизация, в свою очередь, градиентный спуск, которые позволяют найти минимум функции потерь и достичь наилучших результатов.

Практическое применение функции потерь заключается в поиске оптимальных значений параметров модели, чтобы минимизировать функцию потерь. Существуют различные методы оптимизации, такие как модификации градиентного спуска. Например, в задачах классификации функция потерь может помочь модели правильно классифицировать объекты, а оптимизация может помочь найти оптимальные значения весов.

Также функция потерь и оптимизация являются неотъемлемой частью обучения нейронных сетей. Они позволяют нейронной сети "учиться" на примерах и находить оптимальные значения весов, чтобы улучшить качество предсказаний.

Функции потерь и оптимизации являются важными концепциями в машинном обучении. Они позволяют оценить и улучшить работу модели, а также найти оптимальные значения параметров. Знание и понимание этих концепций поможет вам стать более эффективным и успешным специалистом в области машинного обучения.

Что можно почитать еще?

1. Линейные модели
2. Функции ошибок регрессии.
3. Функции потерь и оптимизации

Используемая литература

1. Полное руководство по линейной регрессии в Scikit-Learn
2. ML: Градиентный метод
3. Стохастический градиентный спуск SGD и алгоритм SAG