Курс "Математика для Data Science"
Модуль 3. Математический анализ

Оптимизация в моделях машинного обучения

Предварительные сведения об оптимизации

При тренировке моделей машинного обучения часто возникает вопрос оптимизации функций нескольких переменных. Рассмотрим этот вопрос с математической точки зрения.

Итак, имеется функция $f : \mathbb{R}^n \to \mathbb{R}$ — функция вектора переменных $x = (x_1, x_2, \ldots, x_n)$ и задача оптимизации

$$f(x) \to \min_x .$$

Мы также предполагаем, что функция f дифференцируема в любой точке и, в частности, имеет в каждой точке градиент $\frac{df}{dx}$, компонентами которого являются частные производные:

$$\frac{df}{dx} = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n} \right).$$

Геометрический смысл градиента в произвольной точке x^0 состоит в том, что при изменении аргумента x по направлению градиента $\frac{df}{dx}(x_0)$ в малой окрестности точки x_0 значение функции f наиболее быстро увеличивается. Соответственно, при движении против направления градиента (как говорят, по антиградиенту) значение функции уменьшается. Именно эту идею использует алгоритм оптимизации, который называется *градиентный спуск*.

Итак, давайте придумаем итеративный алгоритм минимизации функции f, то есть построим последовательность точек $\{x^k\}_{k=1}^{\infty}$, для которой x^k всё ближе подходит к минимуму функции. Здесь не следует путать верхний индекс со степенью числа: имеем $x^k = (x_1^k, x_2^k, \ldots, x_n^k)$.

Зафиксируем параметр $\lambda > 0$ — обычно его называют learning rate.

1. Начинаем с некоторой точки x^1.

2. Каждая следующая точка x^{k+1} определяется по формуле

$$x^{k+1} = x^k - \lambda \frac{df}{dx}(x^k).$$

Если расписать тождество выше покомпонентно, то получится формула обновления каждой компоненты вектора x:

$$x_i^{k+1} = x_i^k - \lambda \frac{\partial f}{dx_i}(x^k).$$

При некоторых теоретических ограничениях на функцию f можно утверждать, что алгоритм градиентного спуска сходится в локальный минимум. Если локальный минимум единственный (например, при выпуклости функции f), то это гарантирует оптимальность найденного алгоритмом решения.

Линейная регрессия

Линейная регрессия — это алгоритм решения задачи регрессии — задачи восстановления целевой переменной y по признакам $(x_1, x_2, \ldots, x_n)$. От задачи классификации задачу регрессии отличает то, что целевая переменная является произвольным действительным числом.

Итак, пусть дана выборка из ℓ объектов $x^1, x^2, \ldots, x^\ell$, каждый из которых описывается вектором признаков: $x^i = (x_1^i, x_2^i, \ldots, x_n^i)$, $i = 1, 2, \ldots, \ell$. Компактно выборка описывается матрицей "объекты-признаки":

$$X = \begin{pmatrix} x_1^1 & x_2^1 & \ldots & x_k^1 \\ x_1^2 & x_2^2 & \ldots & x_k^2 \\ \ldots & \ldots & \ldots & \ldots \\ x_1^\ell & x_2^\ell & \ldots & x_k^\ell \end{pmatrix}.$$

Для примера можно считать, что мы решаем задачу предсказания стоимости квартиры по её признакам. Тогда $x^i = (x_1^i, x_2^i, \ldots, x_n^i)$ — набор характеристик i-ой квартиры: скажем, площадь, количество комнат, ...

Также дан набор ответов: вектор $y = (y^1, y^2, \ldots, y^\ell)$, где y^i — правильный ответ для объекта x^i. Для нашей задачи это стоимость i-ой квартиры.

Алгоритм линейной регрессии восстанавливает искомую зависимость в виде $a(x) = \langle x, w\rangle$ (так обозначается скалярное произведение векторов x и w), где $w \in \mathbb{R}^n$ — вектор весов алгоритма. Коэффициенты w находятся из *решения задачи оптимизации квадратичной функции потерь*:

$$\sum_{i=1}^{\ell} \big(\langle x^i, w\rangle - y^i\big)^2 \to \min_w \tag{1}$$

Прежде чем предъявить решение данной задачи в явном виде, перепишем её в матричном виде. Положим

$$X = \begin{pmatrix} x_1^1 & x_2^1 & \ldots & x_k^1 \\ x_1^2 & x_2^2 & \ldots & x_k^2 \\ \ldots & \ldots & \ldots & \ldots \\ x_1^\ell & x_2^\ell & \ldots & x_k^\ell \end{pmatrix}, \; w = \begin{pmatrix} w_1 \\ w_2 \\ \ldots \\ w_k \end{pmatrix}$$

— соответственно матрица "объекты-признаки": в строчках записаны векторы признаков каждого объекта, и вектор-столбец весов алгоритма.

Тогда рассмотрим произведение Xw по правилам перемножения матриц. Имеем

$$Xw = \begin{pmatrix} x_1^1 & x_2^1 & \ldots & x_k^1 \\ x_1^2 & x_2^2 & \ldots & x_k^2 \\ \ldots & \ldots & \ldots & \ldots \\ x_1^\ell & x_2^\ell & \ldots & x_k^\ell \end{pmatrix} \times \begin{pmatrix} w_1 \\ w_2 \\ \ldots \\ w_k \end{pmatrix} = \begin{pmatrix} \langle x^1, w\rangle \\ \langle x^2, w\rangle \\ \ldots \\ \langle x^\ell, w\rangle \end{pmatrix}.$$

Наконец, имеем

$$Xw - y = \begin{pmatrix} \langle x^1, w\rangle - y^1 \\ \langle x^2, w\rangle - y^2 \\ \cdots \\ \langle x^\ell, w\rangle - y^\ell \end{pmatrix}.$$

Таким образом, в компонентах вектора $Xw-y$ записаны значения отклонений алгоритма от правильных ответов. Тогда по теореме Пифагора квадрат длины вектора $Xw - y$ равен сумме квадратов отклонений или, иными словами,

$$||Xw + b - y||^2 = \sum_{i=1}^{\ell} \big(\langle x^i, w\rangle - y^i\big)^2,$$

что в точности совпадает с выражением (1). Таким образом, задача минимизации переписывается в компактном виде

$$||Xw - y||^2 \to \min_w. \tag{2}$$

Получившуюся задачу минимизации называют *задачей наименьших квадратов*.

Решим эту задачу оптимизации.

<лекционный материал>

Логистическая регрессия

Логистическая регрессия — это алгоритм линейной классификации, то есть алгоритм классификации на классы $\{+1, -1\}$, имеющий вид $a(x) = \text{sign}(\langle w, x\rangle)$, где

$$\text{sign}(x) = \begin{cases} +1, & x \geqslant 0; \\ -1, & x < 0. \end{cases}$$

Как и в алгоритме линейной регрессии, мы можем считать, что свободный член b тождественно равен 0.

Отличительной особенностью логистической регрессии является то, что основным объектом рассмотрения является не класс объекта, а *вероятность принадлежности объекта классу* $+1$. Искомая вероятность в модели логистической регрессии вычисляется по формуле

$$\text{P}_w(y(x) = +1) = \sigma(\langle x, w\rangle),$$

где

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

Очевидно, вероятность принадлежности объекта классу -1 высчитывается как

$$1 - \text{P}_w(y(x^i) = +1).$$

Итак, необходимо подобрать такие коэффициенты w, чтобы вероятность $\text{P}_w(y(x^i) = +1)$ была как можно больше на объектах, для которых $y^i = +1$, а вероятность $1 - \text{P}_w(y(x^i) = +1)$ — как можно больше на объектах, для которых $y^i = -1$.

Тогда логично записать произведение максимизируемых вероятностей по всем объектам обучающей выборки и максимизировать его по вектору весов w. Таким образом, имеем задачу оптимизации

$$\prod_{y^i=+1} \mathrm{P}_w(y(x^i)=+1) \cdot \prod_{y^i=-1} \big(1-\mathrm{P}_w(y(x^i)=+1)\big) \to \max_w.$$

Оптимизировать произведение большого количества сомножителей сложно, поэтому логарифмируем произведение и будем оптимизировать сумму логарифмов. Также домножим обе части на -1, чтобы иметь задачу *минимизации* (для единообразия с задачей линейной регрессии). Имеем

$$\sum_{y^i=+1} -\ln \mathrm{P}_w(y(x^i)=+1) + \sum_{y^i=-1} -\ln\big(1-\mathrm{P}_w(y(x^i)=+1)\big) \to \min_w. \tag{3}$$

Преобразуем выражение $-\ln(P_w(y(x)=+1))$ для объектов класса $+1$. Имеем

$$-\ln(P_w(y(x^i)=+1)) = -\ln\left(\frac{1}{1+e^{-\langle w,x^i\rangle}}\right) = \ln\left(1+e^{-\langle w,x^i\rangle}\right) = \ln\left(1+e^{-y^i\langle w,x^i\rangle}\right).$$

Здесь мы используем то, что $y^i=+1$. Аналогично, если $y^i=-1$, то

$$-\ln(P_w(y(x^i)=-1)) = \ln\left(1+e^{\langle w,x^i\rangle}\right) = \ln\left(1+e^{-y^i\langle w,x^i\rangle}\right).$$

Таким образом, задачу (3) можно переписать в виде

$$\sum_{i=1}^{\ell} \ln\left(1+e^{-y^i\langle w,x^i\rangle}\right) \to \min_w. \tag{4}$$

Определение. Функция $\ln\left(1+e^{-y\langle w,x\rangle}\right)$ называется *логистической функцией потерь*.

Определение. *Логистическая регрессия* — это алгоритм линейной классификации, который находит вектор параметров w, который является решением задачи минимизации (4).

Замечание. Задача минимизации логистической функции потерь не имеет аналитического решения, в отличие, например, от задачи минимизации квадратичной функции потерь в задаче линейной регрессии. Для решения этой задачи используют метод стохастического градиентного спуска.

Вычислим градиент логистической функции потерь (на i-ом объекте) по переменной w. Имеем

$$\frac{d\left(\ln\left(1+e^{-y\langle w,x\rangle}\right)\right)}{dw} = \frac{1}{1+e^{-y\langle w,x\rangle}} \cdot \frac{d\left(1+e^{-y\langle w,x\rangle}\right)}{dw} = \frac{e^{-y\langle w,x\rangle}}{1+e^{-y\langle w,x\rangle}} \cdot xy = \frac{1}{1+e^{y\langle w,x\rangle}} \cdot xy.$$

Шаг градиентного спуска (по одному объекту x) записывается в виде

$$w^{k+1} = w^k - \lambda \frac{xy}{1+e^{y\langle w,x\rangle}}.$$

Нейронная сеть

В этом разделе мы сконцентрируемся на описании алгоритма обратного распространения ошибки (Back Propagation) для обучения нейронной сети. В качестве примера возьмём полносвязную нейронную сеть из двух слоёв для решения задачи классификации.

Предположим, что мы, как и раньше, решаем задачу предсказания стоимости квартиры. Будем для простоты считать, что у каждой квартиры ровно два признака: площадь (x_1) и количество комнат (x_2). Опишем модель нейронной сети для решения этой задачи.

Итак, нейронная сеть представляет собой набор последовательно применённых "слоёв", каждый из которых состоит из нескольких нейронов. Внутри нейронной сети заключены *параметры нейронной сети*, которые будут настраиваться в процессе обучения.

- Нулевой слой — входной — содержит 2 входных нейрона — по количество признаков каждого объекта. Вход нейронной сети обозначим через $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$.

- Первый слой нейронной сети применяет ко входам линейное преобразование. Такой слой называется *полносвязным*. Для простоты предположим, что на выходе первого слоя также будет 2 нейрона. Их выходы обозначим через $x^1 = \begin{pmatrix} x_1^1 \\ x_2^1 \end{pmatrix}$. При этом вектор x^1 вычисляется по формуле

  $$x^1 = W^1 x + b^1,$$

  где W^1 — **матрица обучаемых параметров** размера 2×2, b^1 — **вектор обучаемых параметров** размера 2. Этот слой перемешивает признаки между собой, формируя два новых комбинированных признака.

- Второй слой представляет из себя нелинейность, применённую поэлементно к выходам первого слоя. В качестве нелинейности часто берут функцию *сигмоиды*, действующей по формуле $\sigma(x) = \frac{1}{1+e^{-x}}$. Формально, выход слоя вычисляется по формуле $x_i^2 = \sigma(x_i^1)$, $i = 1, 2$.

- Третий слой также является полносвязным слоем. При этом на выходе этого слоя будет всего один нейрон, то есть третий слой комбинирует признаки со второго слоя в одно единственное число y, вычисляющееся по формуле $y = W^2 x^2 + b^2$, где W^2 — матрица размера 1×2 (то есть по сути просто вектор), а b^2 — число.

Итак, y является выходом нейронной сети — действительным числом, которое мы интерпретируем как стоимость квартиры с признаками x^0.

Как и в задаче линейной регрессии, мы применяем квадратичную функцию потерь к выходу y, то есть наша цель — минимизация среднего значения квадрата отклонения

$$L = \frac{1}{|X_{\text{train}}|} \sum_{x \in X_{\text{train}}} |y(x) - y_{\text{true}}(x)|^2 \to \min_{W^1, b^1, W_2, b_2}.$$

Для оптимизации нашей функции потерь мы будем использовать метод градиентного спуска. Поскольку оптимизируемые переменные являются параметрами нашей модели, нам необходимо вычислить градиент функции потерь L по переменным W^1, b^1, W_2, b_2. Этим мы и займёмся.

Дифференцирование сложной функции нескольких переменных

Нейронная сеть представляет собой композицию нескольких функций, каждая из которых является дифференцируемой функцией нескольких переменных. Опишем правило вычисления производной функции, которая является сама является сложной функцией нескольких переменных.

Пусть даны функции $u(x, y)$ и $v(x, y)$, дифференцируемые в точке (x_0, y_0), а также функция $f(u, v)$, дифференцируемая в точке $(u(x_0, y_0), v(x_0, y_0))$. Мы хотим научиться вычислять градиент $(f'_x(x_0, y_0), f'_y(x_0, y_0))$.

Теорема 1. *При выполнении условий выше функция $f(u(x, y), v(x, y))$ является дифференцируемой в точке (x_0, y_0), причём выполняются соотношения*

$$f'_x = \frac{\partial f}{\partial x} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial x} + \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial x};$$
$$f'_y = \frac{\partial f}{\partial y} = \frac{\partial f}{\partial u} \cdot \frac{\partial u}{\partial y} + \frac{\partial f}{\partial v} \cdot \frac{\partial v}{\partial y}.$$

Иными словами,

$$\begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial v}{\partial x} \\ \frac{\partial u}{\partial y} & \frac{\partial v}{\partial y} \end{pmatrix} \begin{pmatrix} \frac{\partial f}{\partial u} \\ \frac{\partial f}{\partial v} \end{pmatrix}.$$

Матрица $\begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{pmatrix}$ называется *матрицей частных производных* или *матрицей Якоби* замены $\{u(x, y), v(x, y)\}$ и обозначается

$$\frac{\partial(u, v)}{\partial(x, y)}.$$

В этих терминах

$$\begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix} = \left(\frac{\partial(u, v)}{\partial(x, y)}\right)^\top \begin{pmatrix} \frac{\partial f}{\partial u} \\ \frac{\partial f}{\partial v} \end{pmatrix}.$$

Полная аналогия с равенством

$$\frac{dy}{dt} = \frac{dy}{dx}\frac{dx}{dt}.$$

1. Пусть функции $u(x, y)$ и $v(x, y)$ являются линейными функциями. Найдите матрицу Якоби такой замены. Иными словами, выясните, как преобразовывается градиент функции потерь под действием линейного оператора.

2. Пусть функции $u(x, y)$ и $v(x, y)$ являются поэлементным применением дифференцируемой функции σ, т.е. $u(x, y) = \sigma(x)$, $v(x, y) = \sigma(y)$. айдите матрицу Якоби такой замены. Иными словами, выясните, как преобразовывается градиент функции потерь под действием поэлементного действия функции активации.

Замечание. В свете всего вышесказанного, можно вычислить градиент функции потерь L по всем переменным нейронной сети. Если же мы имеем эффективный механизм вычисления градиента функции потерь, то это автоматически является механизмом обучения нейронной сети, что мы и хотели получить.

3. Выпишите в явном виде градиенты функции потерь по всем переменным в нейронной сети из примера.