Лекция 3
Линейная регрессия

Е. А. Соколов
ФКН ВШЭ

22 сентября 2021 г.

1 Переобучение

Нередко в машинном обучении модель оказывается *переобученной* — её качество на новых данных существенно хуже качества на обучающей выборке. Действительно, при обучении мы требуем от модели лишь хорошего качества на обучающей выборке, и совершенно не очевидно, почему она должна при этом хорошо *обобщать* эти результаты на новые объекты.

В следующем разделе мы обсудим подходы к оцениванию обобщающей способности, а пока разберём явление переобучения на простом примере. Рассмотрим некоторую одномерную выборку, значения единственного признака x в которой генерируются равномерно на отрезке $[0, 1]$, а значения целевой переменной выбираются по формуле $y = \cos(1.5\pi x) + \mathcal{N}(0, 0.01)$, где $\mathcal{N}(\mu, \sigma^2)$ — нормальное распределение со средним μ и дисперсией σ^2. Попробуем восстановить зависимость с помощью линейных моделей над тремя наборами признаков: $\{x\}$, $\{x, x^2, x^3, x^4\}$ и $\{x, x^2, \ldots, x^{15}\}$. Соответствующие результаты представлены на рис. 1.

Видно, что при использовании признаков высоких степеней модель получает возможность слишком хорошо подстроиться под выборку, из-за чего становится непригодной для дальнейшего использования. Эту проблему можно решать многими способами — например, использовать более узкий класс моделей или штрафовать за излишнюю сложность полученной модели. Так, можно заметить, что у переобученной модели, полученной на третьем наборе признаков, получаются очень большие коэффициенты при признаках. Как правило, именно норма вектора коэффициентов используется как величина, которая штрафуется для контроля сложности модели. Такой подход называется *регуляризацией*, речь о нём пойдёт в следующих лекциях.

2 Оценивание качества моделей

В примере, о котором только что шла речь, мы не можем обнаружить переобученность модели по обучающей выборке [1]. С другой стороны, если бы у нас были

[1] Конечно, это можно было бы заметить по большим весам в модели, но связь между нормой весов и обобщающей способностью алгоритма неочевидна.

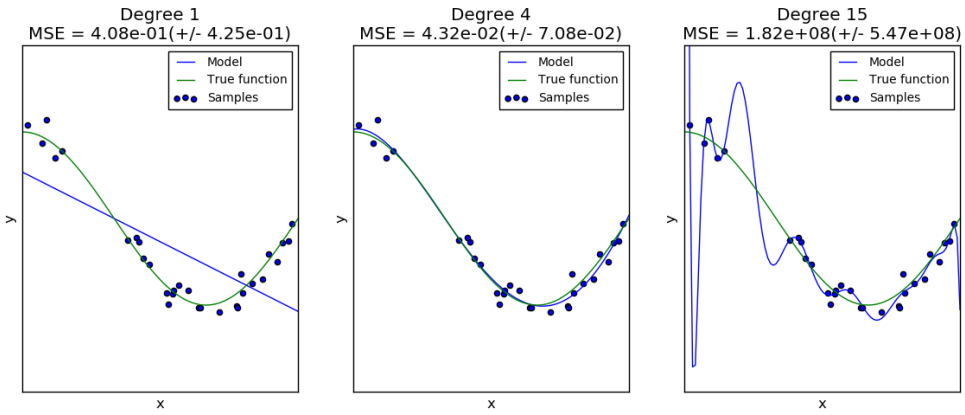

Рис. 1. Регрессионные кривые для признаковых наборов различной сложности.

дополнительные объекты с известными ответами, то по ним заметить низкое качество модели было бы довольно легко.

На данной идее основан подход с *отложенной выборкой*. Имеющиеся размеченные данные (т.е. данные с известными ответами) разделяются на две части: обучающую и контрольную. На обучающей выборке, как это следует из названия, модель обучается, а на контрольной выборке проверяется её качество. Если значение функционала на контрольной выборке оказалось удовлетворительным, то можно считать, что модель смогла извлечь закономерности при обучении.

Использование отложенной выборки приводит к одной существенной проблеме: результат существенно зависит от конкретного разбиения данных на обучение и контроль. Мы не знаем, какое качество получилось бы, если бы объекты из данного контроля оказались в обучении. Решить эту проблему можно с помощью *кросс-валидации*. Размеченные данные разбиваются на k блоков $X_1, \ldots, X_k$ примерно одинакового размера. Затем обучается k моделей $a_1(x), \ldots, a_k(x)$, причём i-я модель обучается на объектах из всех блоков, кроме блока i. После этого качество каждой модели оценивается по тому блоку, который не участвовал в её обучении, и результаты усредняются:

$$\mathrm{CV} = \frac{1}{k} \sum_{i=1}^{k} Q\left(a_i(x), X_i\right).$$

Допустим, мы оценили качество некоторого метода обучения с помощью кросс-валидации и убедились, что выдаваемые им модели хорошо обобщают. Как получить финальную модель для дальнейшего использования? Разумными будут следующие варианты:

1. Обучить модель тем же способом на всех доступных данных. Если мы использовали кросс-валидацию, скажем, по 5 блокам, то каждая модель обучалась на 80% от общего числа объектов обучающей выборки. Если построить итоговую модель на всей выборке, то её параметры будут подобраны по большему числу объектов и можно надеяться, что качество вырастет.

2. Если возможности обучить финальную модель нет (например, это слишком долго), то можно построить *композицию* из моделей $a_1(x), \ldots, a_k(x)$, получен-

ных в процессе кросс-валидации. Под композицией может пониматься, например, усреднение прогнозов этих моделей, если мы решаем задачу регрессии. Позже в курсе мы выясним, что идея такого объединения нескольких моделей оказывается полезной при правильном применении.

3 Обучение линейной регрессии

Нередко линейная регрессия обучается с использованием среднеквадратичной ошибки. В этом случае получаем задачу оптимизации (считаем, что среди признаков есть константный, и поэтому свободный коэффициент не нужен):

$$\frac{1}{\ell}\sum_{i=1}^{\ell}\left(\langle w, x_i\rangle - y_i\right)^2 \to \min_w$$

Эту задачу можно переписать в матричном виде. Если X — матрица «объекты-признаки», y — вектор ответов, w — вектор параметров, то приходим к виду

$$\frac{1}{\ell}\left\|Xw - y\right\|^2 \to \min_w, \tag{3.1}$$

где используется обычная L_2-норма. Если продифференцировать данный функционал по вектору w, приравнять к нулю и решить уравнение, то получим явную формулу для решения (подробный вывод формулы можно найти в материалах семинаров):

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

Безусловно, наличие явной формулы для оптимального вектора весов — это большое преимущество линейной регрессии с квадратичным функционалом. Но данная формула не всегда применима по ряду причин:

- Обращение матрицы — сложная операция с кубической сложностью от количества признаков. Если в выборке тысячи признаков, то вычисления могут стать слишком трудоёмкими. Решить эту проблему можно путём использования численных методов оптимизации.

- Матрица $X^T X$ может быть вырожденной или плохо обусловленной. В этом случае обращение либо невозможно, либо может привести к неустойчивым результатам. Проблема решается с помощью регуляризации, речь о которой пойдёт ниже.

Следует понимать, что аналитические формулы для решения довольно редки в машинном обучении. Если мы заменим MSE на другой функционал, то найти такую формулу, скорее всего, не получится. Желательно разработать общий подход, в рамках которого можно обучать модель для широкого класса функционалов. Такой подход действительно есть для дифференцируемых функций — обсудим его подробнее.

4 Градиентный спуск и оценивание градиента

Оптимизационные задачи вроде (3.1) можно решать итерационно с помощью градиентных методов (или же методов, использующих как градиент, так и информацию о производных более высокого порядка).

Градиентом функции $f : \mathbb{R}^d \to \mathbb{R}$ называется вектор его частных производных:

$$\nabla f(x_1, \ldots, x_d) = \left(\frac{\partial f}{\partial x_j}\right)_{j=1}^{d}.$$

Известно, что градиент является направлением наискорейшего роста функции, а антиградиент (т.е. $-\nabla f$) — направлением наискорейшего убывания. Это ключевое свойство градиента, обосновывающее его использование в методах оптимизации.

Докажем данное утверждение. Пусть $v \in \mathbb{R}^d$ — произвольный вектор, лежащий на единичной сфере: $\|v\| = 1$. Пусть $x_0 \in \mathbb{R}^d$ — фиксированная точка пространства. Скорость роста функции в точке x_0 вдоль вектора v характеризуется производной по направлению $\frac{\partial f}{\partial v}$:

$$\frac{\partial f}{\partial v} = \frac{d}{dt} f(x_{0,1} + tv_1, \ldots, x_{0,d} + tv_d)|_{t=0}.$$

Из курса математического анализа известно, что данную производную сложной функции можно переписать следующим образом:

$$\frac{\partial f}{\partial v} = \sum_{j=1}^{d} \frac{\partial f}{\partial x_j} \frac{d}{dt}(x_{0,j} + tv_j) = \sum_{j=1}^{d} \frac{\partial f}{\partial x_j} v_j = \langle \nabla f, v \rangle.$$

Распишем скалярное произведение:

$$\langle \nabla f, v \rangle = \|\nabla f\| \|v\| \cos\varphi = \|\nabla f\| \cos\varphi,$$

где φ — угол между градиентом и вектором v. Таким образом, производная по направлению будет максимальной, если угол между градиентом и направлением равен нулю, и минимальной, если угол равен 180 градусам. Иными словами, производная по направлению максимальна вдоль градиента и минимальна вдоль антиградиента.

У градиента есть ещё одно свойство, которое пригодится нам при попытках визуализировать процесс оптимизации, — он ортогонален линиям уровня. Докажем это. Пусть x_0 — некоторая точка, $S(x_0) = \{x \in \mathbb{R}^d \mid f(x) = f(x_0)\}$ — соответствующая линия уровня. Разложим функцию в ряд Тейлора на этой линии в окрестности x_0:

$$f(x_0 + \varepsilon) = f(x_0) + \langle \nabla f, \varepsilon \rangle + o(\|\varepsilon\|),$$

где $x_0 + \varepsilon \in S(x_0)$. Поскольку $f(x_0 + \varepsilon) = f(x_0)$ (как-никак, это линия уровня), получим

$$\langle \nabla f, \varepsilon \rangle = o(\|\varepsilon\|).$$

Поделим обе части на $\|\varepsilon\|$:

$$\left\langle \nabla f, \frac{\varepsilon}{\|\varepsilon\|} \right\rangle = o(1).$$

Устремим $\|\varepsilon\|$ к нулю. При этом вектор $\frac{\varepsilon}{\|\varepsilon\|}$ будет стремится к касательной к линии уровня в точке x_0. В пределе получим, что градиент ортогонален этой касательной.

§4.1 Градиентный спуск

Основное свойство антиградиента — он указывает в сторону наискорейшего убывания функции в данной точке. Соответственно, будет логично стартовать из некоторой точки, сдвинуться в сторону антиградиента, пересчитать антиградиент и снова сдвинуться в его сторону и т.д. Запишем это более формально. Пусть $w^{(0)}$ — начальный набор параметров (например, нулевой или сгенерированный из некоторого случайного распределения). Тогда градиентный спуск состоит в повторении следующих шагов до сходимости:

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla Q(w^{(k-1)}). \tag{4.1}$$

Здесь под $Q(w)$ понимается значение функционала ошибки для набора параметров w.

Через η_k обозначается длина шага, которая нужна для контроля скорости движения. Можно делать её константной: $\eta_k = c$. При этом если длина шага слишком большая, то есть риск постоянно «перепрыгивать» через точку минимума, а если шаг слишком маленький, то движение к минимуму может занять слишком много итераций. Иногда длину шага монотонно уменьшают по мере движения — например, по простой формуле

$$\eta_k = \frac{1}{k}.$$

В пакете `vowpal wabbit`, реализующем настройку и применение линейных моделей, используется более сложная формула для шага в градиентном спуске:

$$\eta_k = \lambda \left(\frac{s_0}{s_0 + k} \right)^p,$$

где λ, s_0 и p — параметры (мы опустили в формуле множитель, зависящий от номера прохода по выборке). На практике достаточно настроить параметр λ, а остальным присвоить разумные значения по умолчанию: $s_0 = 1$, $p = 0.5$, $d = 1$.

TODO: при грамотном подборе шага и липшицевости градиента гарантируется сходимость к локальному минимуму

Останавливать итерационный процесс можно, например, при близости градиента к нулю или при слишком малом изменении вектора весов на последней итерации.

Если функционал $Q(w)$ выпуклый, гладкий и имеет минимум w^*, то имеет место следующая оценка сходимости:

$$Q(w^{(k)}) - Q(w^*) = O(1/k).$$

§4.2 Оценивание градиента

Как правило, в задачах машинного обучения функционал $Q(w)$ представим в виде суммы ℓ функций:

$$Q(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} q_i(w).$$

В таком виде, например, записан функционал в задаче (3.1), где отдельные функции $q_i(w)$ соответствуют ошибкам на отдельных объектах.

Проблема метода градиентного спуска (4.1) состоит в том, что на каждом шаге необходимо вычислять градиент всей суммы (будем его называть полным градиентом):

$$\nabla_w Q(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} \nabla_w q_i(w).$$

Это может быть очень трудоёмко при больших размерах выборки. В то же время точное вычисление градиента может быть не так уж необходимо — как правило, мы делаем не очень большие шаги в сторону антиградиента, и наличие в нём неточностей не должно сильно сказаться на общей траектории. Опишем несколько способов оценивания полного градиента.

Оценить градиент суммы функций можно градиентом одного случайно взятого слагаемого:

$$\nabla_w Q(w) \approx \nabla_w q_{i_k}(w),$$

где i_k — случайно выбранный номер слагаемого из функционала. В этом случае мы получим метод *стохастического градиентного спуска* (stochastic gradient descent, SGD) [1]:

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla q_{i_k}(w^{(k-1)}).$$

Для выпуклого и гладкого функционала может быть получена следующая оценка:

$$\mathbb{E}\left[Q(w^{(k)}) - Q(w^*)\right] = O(1/\sqrt{k}).$$

Таким образом, метод стохастического градиента имеет менее трудоемкие итерации по сравнению с полным градиентом, но и скорость сходимости у него существенно меньше.

TODO: константная длина шага не подойдёт

Отметим одно важное преимущество метода стохастического градиентного спуска. Для выполнения одного шага в данном методе требуется вычислить градиент лишь одного слагаемого — а поскольку одно слагаемое соответствует ошибке на одном объекте, то получается, что на каждом шаге необходимо держать в памяти всего один объект из выборки. Данное наблюдение позволяет обучать линейные модели на очень больших выборках: можно считывать объекты с диска по одному, и по каждому делать один шаг метода SGD.

Можно повысить точность оценки градиента, используя несколько слагаемых вместо одного:

$$\nabla_w Q(w) \approx \frac{1}{n} \sum_{j=1}^{n} \nabla_w q_{i_{kj}}(w),$$

где i_{kj} — случайно выбранные номера слагаемых из функционала (j пробегает значения от 1 до n), а n — параметр метода, размер пачки объектов для одного градиентного шага. С такой оценкой мы получим метод mini-batch gradient descent, который часто используется для обучения дифференцируемых моделей.

В 2013 году был предложен метод *среднего стохастического градиента* (stochastic average gradient) [2], который в некотором смысле сочетает низкую сложность итераций стохастического градиентного спуска и высокую скорость сходимости полного градиентного спуска. В начале работы в нём выбирается первое приближение w^0, и инициализируются вспомогательные переменные z_i^0, соответствующие градиентам слагаемых функционала:

$$z_i^{(0)} = \nabla q_i(w^{(0)}), \qquad i = 1, \ldots, \ell.$$

На k-й итерации выбирается случайное слагаемое i_k и обновляются вспомогательные переменные:

$$z_i^{(k)} = \begin{cases} \nabla q_i(w^{(k-1)}), & \text{если } i = i_k; \\ z_i^{(k-1)} & \text{иначе.} \end{cases}$$

Иными словами, пересчитывается один из градиентов слагаемых. Оценка градиента вычисляется как среднее вспомогательных переменных — то есть мы используем все слагаемые, как в полном градиенте, но при этом почти все слагаемые берутся с предыдущих шагов, а не пересчитываются:

$$\nabla_w Q(w) \approx \frac{1}{\ell} \sum_{i=1}^{\ell} z_i^{(k)}.$$

Наконец, делается градиентный шаг:

$$w^{(k)} = w^{(k-1)} - \eta_k \frac{1}{\ell} \sum_{i=1}^{\ell} z_i^{(k)}.$$

Данный метод имеет такой же порядок сходимости для выпуклых и гладких функционалов, как и обычный градиентный спуск:

$$\mathbb{E}\left[Q(w^{(k)}) - Q(w^*)\right] = O(1/k).$$

Существует множество других способов получения оценки градиента. Например, это можно делать без вычисления каких-либо градиентов вообще [3] — достаточно взять случайный вектор u на единичной сфере и домножить его на значение функции в данном направлении:

$$\nabla_w Q(w) = Q(w + \delta u)u.$$

Можно показать, что данная оценка является несмещённой для сглаженной версии функционала Q.

В задаче оценивания градиента можно зайти ещё дальше. Если вычислять градиенты $\nabla_w q_i(w)$ сложно, то можно *обучить модель*, которая будет выдавать оценку градиента на основе текущих значений параметров. Этот подход был предложен для обучения глубинных нейронных сетей [4].

§4.3 Модификации градиентного спуска

С помощью оценок градиента можно уменьшать сложность одного шага градиентного спуска, но при этом сама идея метода не меняется — мы движемся в сторону наискорейшего убывания функционала. Конечно, такой подход не идеален, и можно по-разному его улучшать, устраняя те или иные его проблемы. Мы разберём два примера таких модификаций — одна будет направлена на борьбу с осцилляциями, а вторая позволит автоматически подбирать длину шага.

Метод импульса (momentum). Может оказаться, что направление антиградиента сильно меняется от шага к шагу. Например, если линии уровня функционала сильно вытянуты, то из-за ортогональности градиента линиям уровня он будет менять направление на почти противоположное на каждом шаге. Такие осцилляции будут вносить сильный шум в движение, и процесс оптимизации займёт много итераций. Чтобы избежать этого, можно усреднять векторы антиградиента с нескольких предыдущих шагов — в этом случае шум уменьшится, и такой средний вектор будет указывать в сторону общего направления движения. Введём для этого вектор инерции:

$$h_0 = 0;$$
$$h_k = \alpha h_{k-1} + \eta_k \nabla_w Q(w^{(k-1)}).$$

Здесь α — параметр метода, определяющей скорость затухания градиентов с предыдущих шагов. Разумеется, вместо вектора градиента может быть использована его аппроксимация. Чтобы сделать шаг градиентного спуска, просто сдвинем предыдущую точку на вектор инерции:

$$w^{(k)} = w^{(k-1)} - h_k.$$

Заметим, что если по какой-то координате градиент постоянно меняет знак, то в результате усреднения градиентов в векторе инерции эта координата окажется близкой к нулю. Если же по координате знак градиента всегда одинаковый, то величина соответствующей координаты в векторе инерции будет большой, и мы будем делать большие шаги в соответствующем направлении.

AdaGrad и RMSprop. Градиентный спуск очень чувствителен к выбору длины шага. Если шаг большой, то есть риск, что мы будем «перескакивать» через точку минимума; если же шаг маленький, то для нахождения минимума потребуется много итераций. При этом нет способов заранее определить правильный размер шага — к тому же, схемы с постепенным уменьшением шага по мере итераций могут тоже плохо работать.

В методе AdaGrad предлагается сделать свою длину шага для каждой компоненты вектора параметров. При этом шаг будет тем меньше, чем более длинные шаги мы делали на предыдущих итерациях:

$$G_{kj} = G_{k-1,j} + (\nabla_w Q(w^{(k-1)}))_j^2;$$
$$w_j^{(k)} = w_j^{(k-1)} - \frac{\eta_t}{\sqrt{G_{kj} + \varepsilon}} (\nabla_w Q(w^{(k-1)}))_j.$$

Здесь ε — небольшая константа, которая предотвращает деление на ноль. В данном методе можно зафксировать длину шага (например, $\eta_k = 0.01$) и не подбирать её в процессе обучения. Отметим, что данный метод подходит для разреженных задач, в которых у каждого объекта большинство признаков равны нулю. Для признаков, у которых ненулевые значения встречаются редко, будут делаться большие шаги; если же какой-то признак часто является ненулевым, то шаги по нему будут небольшими.

У метода AdaGrad есть большой недостаток: переменная G_{kj} монотонно растёт, из-за чего шаги становятся всё медленнее и могут остановиться ещё до того, как достигнут минимум функционала. Проблема решается в методе RMSprop, где используется экспоненциальное затухание градиентов:

$$G_{kj} = \alpha G_{k-1,j} + (1-\alpha)(\nabla_w Q(w^{(k-1)}))_j^2.$$

В этом случае размер шага по координате зависит в основном от того, насколько быстро мы двигались по ней на последних итерациях.

Список литературы

[1] *Robbins, H., Monro S.* (1951). A stochastic approximation method. // Annals of Mathematical Statistics, 22 (3), p. 400-407.

[2] *Schmidt, M., Le Roux, N., Bach, F.* (2013). Minimizing finite sums with the stochastic average gradient. // Arxiv.org.

[3] *Flaxman, Abraham D. and Kalai, Adam Tauman and McMahan, H. Brendan* (2005). Online Convex Optimization in the Bandit Setting: Gradient Descent Without a Gradient. // Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms.

[4] *Jaderberg, M. et. al* (2016). Decoupled Neural Interfaces using Synthetic Gradients. // Arxiv.org.