Лекция 2. Визуализация данных. Математические модели и методы.

Визуализация - задача изображения многомерных объектов в двумерном или трехмерном пространстве таким образом, что сохранялось как можно больше зависимостей и отношений между ними.

Так, в методе многомерного шкалирования (multidimensional scaling, MDS) минимизируются квадраты отклонений между исходными и новыми попарными расстояниями:

$$\sum_{i \neq j}^{l} \left(\rho(x_i, x_j) - \rho(z_i, z_j)\right)^2 \to \min_{z_1,\ldots,z_l},$$

где $x_i \in R^D -$ исходные объекты, а $z_i \in R^d, 2 \leq d \leq 3 -$ их низкоразмерные проекции.

Обратим внимание на две особенности данного подхода:

• Исходные объекты не обязаны принадлежать евклидову пространству – достаточно лишь уметь вычислять расстояния между ними. Благодаря этому можно визуализировать даже сложные объекты вроде строк.

• Проекции объектов ищутся непосредственно, без какой-либо параметрической зависимости между ними и исходными представлениями объектов. Из-за этого затруднительно добавить к визуализации новые данные. Впрочем, это и не нужно – мы ведь хотим просто нарисовать объекты и посмотреть на них. Если же требуется отображать и новые, тестовые данные, то следует пользоваться методами понижения размерности, которые преобразуют объекты с помощью некоторой модели.

Наивный байесовский классификатор

Наивный байесовский классификатор - вероятностный классификатор, основанный на теореме Байеса. Он называется наивным, потому как считает, что наличие одного из признаков никак не зависит от наличия другого. С первого взгляда, такое допущение может показаться слишком сильным, но на практике этот алгоритм показывает достаточно хорошие результаты.

Теорема Байеса говорит о том, как рассчитать апостериорную вероятность:

$$P(c_i|x) = \frac{P(x|c_i) * P(c_i)}{P(x)}$$

Здесь $P(c_i|x) -$ вероятность того, что вектор $x \in X$ относится к классу $c_i \in C$;

$P(x|c_i) -$ вероятность того, что x примет данное значение, при условии, что выбран класс c_i; $P(c_i) -$ априорная вероятность данного класса; $P(x) -$ априорная вероятность данного вектора.

Для того, чтобы понять, какой у вектора x класс, найдем такой класс, который бы максимизировал $P(c_i|x)$.

$$y_i = \arg\max_{c_i \in C} P(c_i|x) = \arg\max_{c_i \in C} \frac{P(x|c_i) * P(c_i)}{P(x)}$$

Можно заметить, что для решения задачи нахождения аргумента, при котором достигается максимум, не нужно учитывать знаменатель, т.к. он не зависит от c_i.

$$y_i = \arg\max_{c_i \in C} P(c_i|x) * P(c_i)$$

Далее, самым интересным в этой формуле является $P(x|c_i)$, распишем его:

$$P(x|c_i) = P(x_1, x_2 \ \dots, x_n|c_i) = P(x_1|c_i) * P(x_2, x_3 \ \dots, x_n|c_i) = P(x_1|c_i) * P(x_2|c_i, x_1) *$$
$$* P(x_3 \ \dots, x_n|c_i, x_1, x_2)$$

Продолжая эту цепочку равенств, можно прийти к следующему:

$$P(x|c_i) = P(x_1|c_i) * P(x_2|c_i, x_1) * \dots * \ P(x_n|c_i, x_1, x_2, \dots, x_{n-1})$$

Эту вероятность достаточно сложно посчитать, т.к. в большинстве случаев непонятно, как именно появление одного признака зависит от появления другого, т.е. $P(x_n|c_i, x_1, x_2, \dots, x_n - 1)$.

Далее, воспользуемся предположением о независимости признаков. Именно в этот момент алгоритм становится “наивным”.

$$P(x_2|c_i, x_1) = P(x_1|c_i)$$

...

$$P(x_n|c_i, x_1, x_2, \dots, x_{n-1}) = P(x_n|c_i)$$

Тогда, если учесть данное предположение, задача приводится к следующему виду:

$$y_i = \arg\max_{c_i \in C} P(c_i) * P(x_1|c_i) * P(x_2|c_i) * \dots * \ P(x_n|c_i)$$

Зная $P(c_i)$ и условные вероятности $P(x_j|c_i)$, можно найти вероятность принадлежности документа x к каждому классу c_i. Далее, тот класс, на котором достигается максимальная вероятность и будет считаться результатом работы алгоритма.

Метод опорных векторов

Метод опорных векторов (англ. Support Vector Machine, SVM) - линейный классификатор, основан на разделении множества векторов из n-мерного пространства гиперплоскостью. Этот метод требует обучения перед классификацией.

Два множества называются линейно разделимыми (или линейно сепарабельными), если существует гиперплоскость, которая отделяет их друг от друга.

Рассмотрим задачу бинарной классификации. Допустим, обучающая выборка является линейно разделимой. Тогда нужно найти такую гиперплоскость, которая бы их разделяла. В общем случае, существует более одной такой гиперплоскости. Возникает вопрос: какую выбрать?

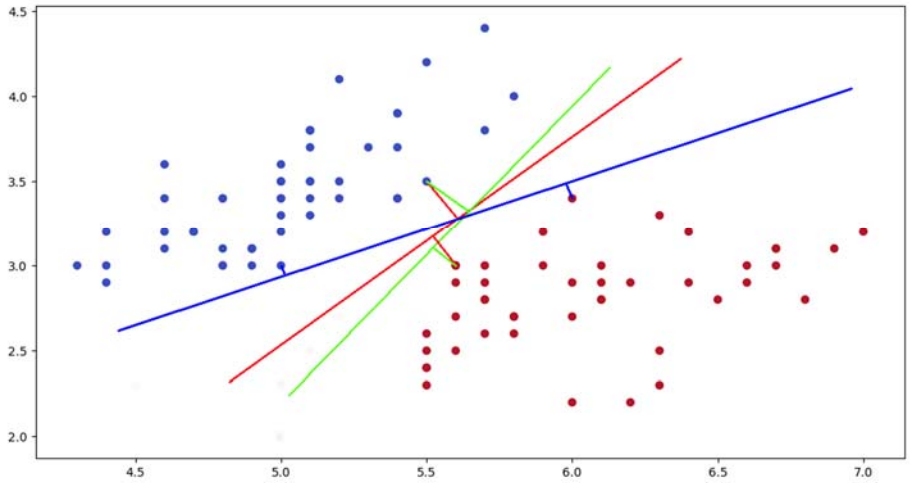

Рисунок 1. Линейно разделимое множество

Логично предположить, что следует выбрать ту гиперплоскость, которая максимально удалена от обоих классов. Такую гиперплоскость и будем называть *оптимальной*.

Допущение о линейной разделимости обучающей выборки было довольно сильным. На практике это скорее исключение, чем правило. Обозначим множество документов за $X \in R^n$, его элементы, соответственно, x_i. Будем рассматривать задачу бинарной классификации, поэтому, не уменьшая общности, обозначим $Y = \{-1, 1\}$.

Множество $S = \{(x_i, y_i) | x_i \in X, y_i \in Y\}, i = 1, \ldots, k$.

Опорными векторами будем называть те вектора, расстояние от которых до разделяющей гиперплоскости минимально.

Из аналитической геометрии известно, что гиперплоскость задается следующим уравнением:

$$(c, x) - d = 0$$

Тогда очевидно, что документы одного класса должны удовлетворять $(c, x_i) >= d$, а другого $(c, x_i) <= d$. Зафиксируем гиперплоскость, тогда параллельные гиперплоскости, содержащие опорные вектора, будут выглядеть таким образом:

$$(c, x) = d + \varepsilon$$
$$(c, x) = d - \varepsilon$$

Отклонение ε одно и тоже в силу того, что расстояние до опорных векторов будет одинаковым, иначе такая гиперплоскость точно не будет оптимальной (очевидный факт, но может быть следует пояснить).

Сделаем нормировку, а именно поделим вектор c и число d на ε.

$$(c, x) - d = 1$$
$$(c, x) - d = -1$$

Для двумерного случая это продемонстрировано на рисунке 2:

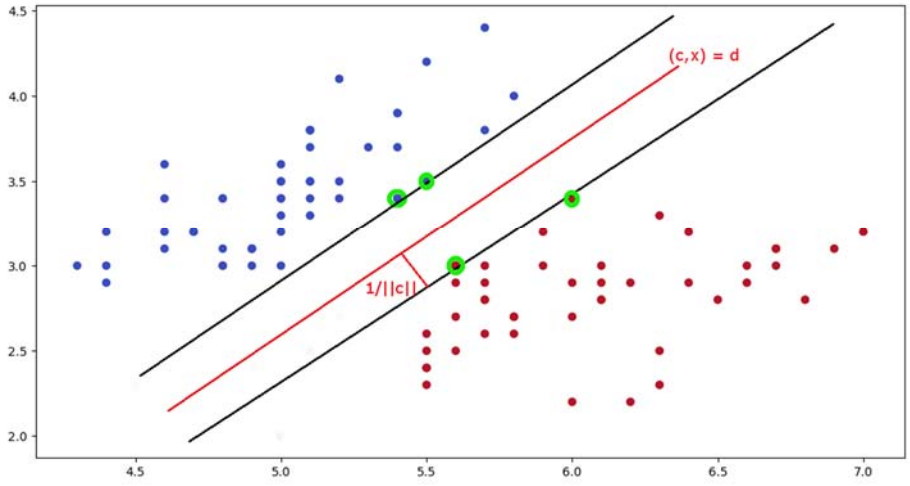

Рисунок 2. Опорные вектора и параллельные гиперплоскости

Введем Евклидову метрику в наше пространство. Тогда расстояние между двумя параллельными гиперплоскостями можно вычислить следующим образом:

$$dist = \frac{|d_1 - d_2|}{\|c\|}$$

Здесь d_1 - свободный член первой гиперплоскости; d_2- свободный член второй гиперплоскости; c - одинаков, вследствие параллельности гиперплоскостей.

Тогда несложно убедиться, что расстояние от параллельных гиперплоскостей до оптимальной гиперплоскости равно $\frac{1}{\|c\|}$. Теперь, чтобы найти оптимальную гиперплоскость, нужно решить следующую задачу оптимизации:

$$\|c\| \to \min$$

$$(c, x_i) - d >= 1, y_i = 1$$
$$(c, x_i) - d <= 1, y_i = -1$$

или в более удобной записи

$$\|c\| \to \min$$
$$y_i * ((c, x_i) - d >= 1$$

Эта задача сводится к двойственной задаче поиска седловой точки функции Лагранжа. В данной работе решение этой задачи описано не будет, т.к. является частью теории методов оптимизации. Все приведенные выше рассуждения были основаны на линейной разделимости обучающей выборки. Что делать в том случае, если это не так? Позволим алгоритму допускать ошибки, но будем искать такую гиперплоскость, которая бы минимизировала их.

Тем самым рассуждения выше сохраняются, за исключением самой задачи оптимизации, которая изменится следующим образом:

$$\|c\| + \sum_{i=o} n\delta_i \to min$$
$$y_i * ((c, x_i) - d >= 1 - \delta_i$$
$$\delta_i >= 0, i = 0, \dots, n$$

Здесь, δ_i равно нулю в том случае, если соответствующее неравенство выполняется и без него. Т.е. δ_i является минимально возможным значением, при котором неравенство будет выполнено. Решая эти задачи, можно получить вектор c, который и нужен для построения гиперплоскости.

Метод K-ближайшего соседа

Метод K-ближайшего соседа (англ.: k-nearest neighbors method, k-NN) – один из методов решения задачи классификации.

Предполагается, что уже имеется какое-то количество объектов с точной классификацией (т.е. для каждого них точно известно, какому классу он принадлежит). Нужно выработать правило, позволяющее отнести новый объект к одному из возможных классов (т.е. сами классы известны заранее).

В основе метода лежит следующее правило: объект считается принадлежащим тому классу, к которому относится большинство его ближайших соседей. Под «соседями» здесь понимаются объекты, близкие к исследуемому в том или ином смысле.

Заметим, что здесь необходимо уметь определять, насколько объекты близки друг к другу, т.е. уметь измерять «расстояние» между объектами. Это не обязательно евклидово

расстояние. Это может быть мера близости объектов, например, по цвету, форме, вкусу, запаху, интересам (если речь идёт о формировании групп людей), особенностям поведения и т.д. Следовательно, для применения метода k- NN в пространстве признаков объектов должна быть введена некоторая метрика (т.е. функция расстояния).

Предполагается, что объекты с близкими значениями одних признаков будут близки и по другим признакам (т.е. относиться к одному и тому же классу).

Рассмотрим работу метода k-NN на простом примере

Таблица 1. Исходные данные

Продукт	Сладость	Хруст	Класс
яблоко	9	8	фрукт
бекон	1	4	протеин
банан	10	1	фрукт
…	…	…	…

Здесь качества продуктов (сладость и хруст) оцениваются по 10-балльной шкале. Эти значения можно рассматривать как координаты точек (продуктов) в 2-мерном пространстве. По оси будем откладывать степень сладости продукта, по оси ординат – степень хруста. Получим график, изображённый на рисунке 5.

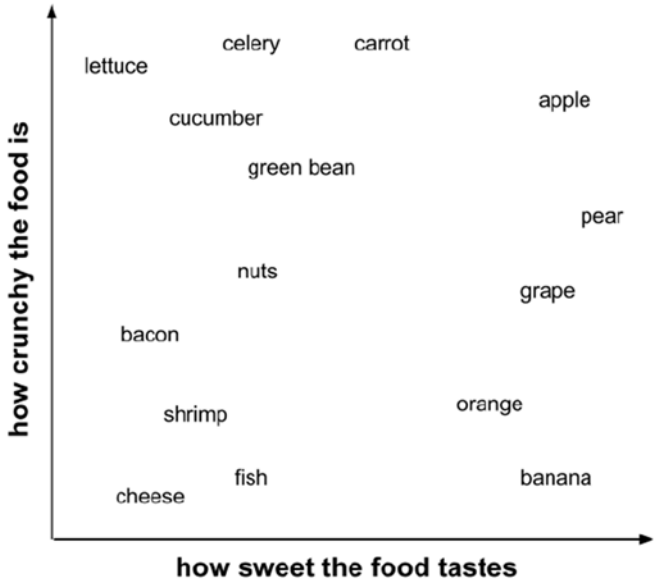

Рисунок 3. Результат визуализации данных

Для каждого из названных продуктов мы точно знаем тип (класс) – см. последний столбец таблицы. Можно заметить, что точки (продукты) на графике можно разбить на классы:

– в левом верхнем углу «группируются» овощи (огурец, морковь, салат-латук, сельдерей) – они хрустящие и несладкие,

– в левом нижнем – продукты, богатые протеином (бекон, креветки, сыр, рыба, орехи) – они не хрустящие и несладкие,

– справа «выстроились» фрукты (яблоко, груша, виноград, апельсин, банан) – они сладкие по сравнению с другими классами, но неоднородны в отношении хруста.

Разбиение на классы показано на рисунке 4.

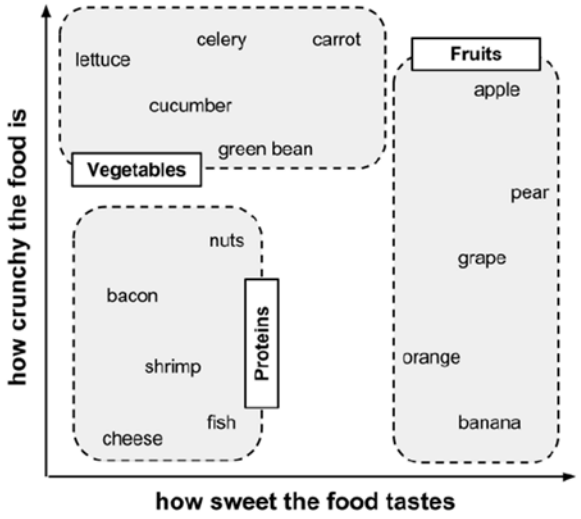

Рисунок 4. Продукты, очевидно, образуют

Предположим теперь, что нам предложен новый продукт, и мы должны определить, к какому классу он относится – см. рисунок 5.

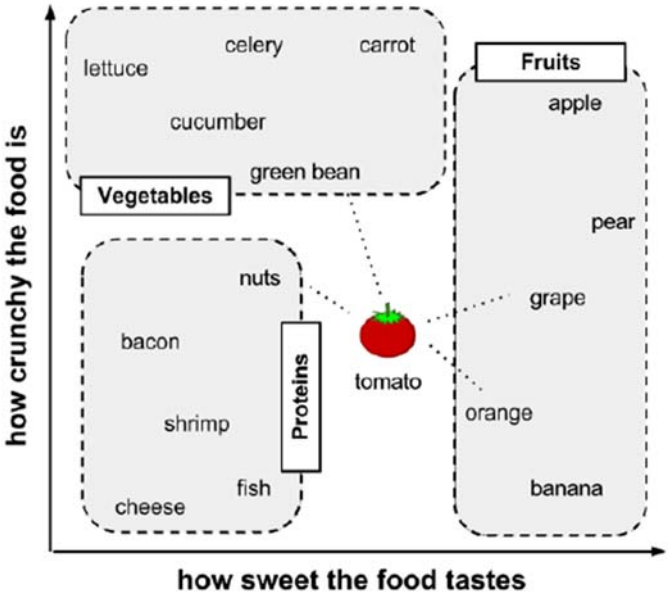

Рисунок 5. Помидор – это овощ или фрукт?

Согласно методу k-NN мы отнесём его к тому классу, к которому принадлежит большинство из k его ближайших соседей. Расстояние между объектами будем понимать в смысле *Евклидовой нормы*, т.е. расстояние между объектами с координатами (x_1, y_1) и (x_2, y_2) равно $\sqrt{(x1 - y1)^2 + (x2 - y2)^2}$.

Так, если у томата показатель сладости равен 3, а показатель хруста равен 7, то его расстояния до яблока, бекона и банана равны примерно 6,1; 3,6, 9,2, соответственно.

Приведём расстояния от томата до всех остальных продуктов, упорядочив их по возрастанию:

Таблица 2. Расчет расстояния между помидорами и его ближайшими соседями

№	Продукт	Класс	Расстояние до томата
1	апельсин	фрукт	1,4
2	виноград	фрукт	2,2
3	креветка	протеин	3,5
4	бекон	протеин	3,6
5	орехи	протеин	3,6
6	сыр	протеин	4,0
7	бобы	протеин	4,2
8	огурец	овощ	5,9
9	яблоко	фрукт	6,1
10	морковь	овощ	6,8
11	сельдерей	овощ	7,0
12	салат-латук	овощ	7,2
13	банан	фрукт	9,2

Теперь нужно выбрать число k и определить, к какому классу принадлежат большинство из k ближайших соседей томата. Так, если $k = 1$, то ближайший сосед – один, и это апельсин, он – фрукт. При $k = 2$ это апельсин и виноград, оба фрукты. При $k = 3$ мы имеем 2 фрукта (апельсин и виноград) и креветку (протеин). Значит, метод k-NN опять даст ответ: «фрукт». При $k = 4$ ответом сбудет «фрукт или про-теин с равной вероятностью». При $k = 5, k = 6, k = 7, k = 8$ «побеждает» протеин. Этот процесс можно продолжать и далее, увеличивая значение k. Мы видим, что *результат, получаемый методом k-NN, сильно зависит от выбора параметра k*.

Возникает вопрос: как выбрать значение параметра k, чтобы минимизировать количество неверных ответов, полученных методом $k - NN$?

Если мы выберем значение k слишком малым, то есть опасность, что единственным ближайшим объектом окажется «выброс», т.е. объект с неправильно определённым классом, и он даст неверное решение. Казалось бы, увеличивая значение

параметра k, мы снижаем вероятность случайного попадания на такие «выбросы» в качестве ближайших соседей исследуемого объекта. Но здесь возникает другая опасность. Чтобы понять в чём она заключается, рассмотрим случай, когда k равно общему числу объектов N. Понятно, что тогда «победит» самый популярный (модальный) класс, и расстояние до исследуемого объекта не будет играть вообще никакой роли. Проблему выбора оптимального значения параметра k называют «bias-variance tradeoff», т.е. «компромисс между «выбросами» и дисперсией». На практике чаще всего полагают $k = [\sqrt{N}]$. Т.е. в нашем примере $k = 3$ и результатом классификации будет то, что помидор – фрукт.

В том случае, если мы уверены в «чистоте» выборки, мы можем выбирать k меньшим. Существует также приём под названием «weighted voting» (т.е. буквально «взвешенное голосование»), при котором более близкие соседи исследуе- мого объекта имеют больший вес, чем более дальние.

Линейные модели

Линейные регрессионные модели, такие модели сводятся к суммированию значений признаков с некоторыми весами:

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j$$

Параметрами модели являются веса или коэффициенты w_j . Вес w_0 также называется свободным коэффициентом или сдвигом (bias). Воспользуемся этим и запишем линейную модель в более компактном виде:

$a(x) = w_0 + \langle w, x \rangle$, где $w = (w_1, \dots, w_d) -$ вектор весов.

Достаточно часто используется следующий приём, позволяющий упростить запись ещё сильнее. Добавим к признаковому описанию каждого объекта $(d + 1) -$ й признак, равный единице. Вес при этом признаке как раз будет иметь смысл свободного коэффициента, и необходимость в слагаемом w_0 отпадёт:

$$a(x) = \langle w, x \rangle.$$

Тем не менее, при такой форме следует соблюдать осторожность и помнить о наличии в выборке специального признака. Например, мы столкнёмся со сложностями, связанными с этим, когда будем говорить о регуляризации.

За счёт простой формы линейные модели достаточно быстро и легко обучаются, и поэтому популярны при работе с большими объёмами данных. Также у них мало

параметров, благодаря чему удаётся контролировать риск переобучения и использовать их для работы с зашумлёнными данными и с небольшими выборками.

Градиентный спуск и оценивание градиента

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

Градиент и его свойства

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

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

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

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

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

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

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.

Покажем теперь, что градиент ортогонален линиям уровня. Пусть x_0 – некоторая точка, $S(x_0) = \{x \in R^d \mid f(x) = f(x_0)\}$ – 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 $\|\varepsilon\|$:

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

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

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:

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

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: $\eta_k = \text{c}$. 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:

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

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:

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

Оценивание градиента

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

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

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

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

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. В этом случае мы получим метод стохастического градиентного спуска (stochastic gradient descent, SGD):

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

где i_k – случайно выбранный номер слагаемого из функционала. Для выпуклого и гладкого функционала может быть получена следующая оценка:

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

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

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.

В 2013 году был предложен метод *среднего стохастического градиента* (stochastic average gradient), 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:

$$z_i^{(0)} = \nabla q_i\left(w^{(0)}\right), \qquad i = 1, \dots, l$$

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

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

Иными словами, пересчитывается один из градиентов слагаемых. Наконец, делается градиентный шаг:

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

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

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

Существует множество других способов получения оценки градиента. Например, это можно делать без вычисления каких-либо градиентов вообще $-$ 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.