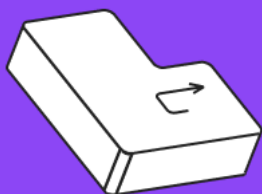


Логистическая регрессия. Градиентный спуск.

Машинное обучение



Оглавление

Введение	2
Термины, используемые в лекции	3
Линейный классификатор	3
Логистическая регрессия	9
Градиентный спуск	15
Практика	22
Что можно почитать еще?	36
Используемая литература	36

Введение

На прошлом занятии мы познакомились с алгоритмом машинного обучения под названием линейная регрессия. Мы разобрали формулу, по которой можно получить веса, выяснили, почему матричный метод не подходит, и обучили модель. Сегодня мы познакомимся с другим типом задач, которые решает машинное обучение. И задача эта называется классификация.

На этой лекции вы найдете ответы на такие вопросы:

- Что такое бинарный линейный классификатор
- Как получить вероятность при классификации
- Что такое градиентный спуск и почему это один из самых мощных алгоритмов поиска минимума функции

Термины, используемые в лекции

Линейный бинарный классификатор – алгоритм для решения задачи классификации, позволяющий определять принадлежность объекта к одному и двух классов.

Логистическая регрессия – алгоритм для решения задачи классификации, позволяющий определять принадлежность и вероятность этой принадлежности объекта к одному из двух классов.

Градиентный спуск – общий метод для обучения моделей в машинном обучении, в основе лежит производная.

Стохастический градиентный спуск – частный метод для обучения моделей в машинном обучении, в основе лежит градиент.

Линейный классификатор

На прошлом занятии мы познакомились с алгоритмом машинного обучения под названием линейная регрессия. Мы разобрали формулу, по которой можно получить веса, выяснили, почему матричный метод не подходит, и обучили модель. Сегодня мы познакомимся с другим типом задач, которые решает машинное обучение. И задача эта называется классификация.

Вы познакомитесь с простейшим классификатором, а затем мы разберем, как добиться от классификатора не только предсказания класса объекта, но и вероятность принадлежности объекта к классу. Затем мы разберем один из самых мощных и используемых алгоритмов для поиска минимума функции, который называется градиентный спуск, и частный случай такого алгоритма. Давайте начнем!

Перед тем, как мы погрузимся в тему, давайте составим небольшой план действий. Когда мы говорили про линейную регрессию, мы сначала определили формулу для предсказаний, затем вывели функцию потерь (это у нас MSE — среднеквадратичная ошибка) для этого алгоритма. Мы сказали, что обучение модели — это почти всегда минимизация какой-то функции. Мы использовали метод наименьших квадратов для минимизации MSE.



Модель линейной регрессии

Формула линейной регрессии:

$$a(x) = w_0 + w_1 x_1 + \dots + w_d x_d$$

Функция потерь, которую мы минимизируем:

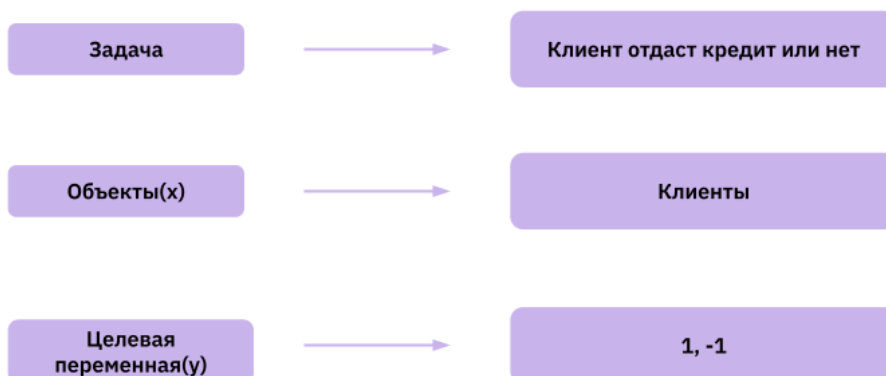
$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

Давайте на этом занятии точно так же, шаг за шагом, определим формулы.

Давайте рассмотрим популярную задачу в банковской системе. Предположим, что мы сотрудники банка, и нам поручили ответственное задание — создать алгоритм, который на основе некоторых данных будет говорить, какой клиент банка отдаст кредит, а какой — не отдаст. Давайте именно так определим нашу задачу и не будем пока ее усложнять.



Пример





Такая постановка задачи относится к области кредитного скоринга. Если раньше в банке сидел специальный человек, который решал, давать вам кредит или нет, то теперь этим занимается система, оценивающая огромное количество разных характеристик человека, основанная как раз на машинном обучении.

Формально, нам нужно придумать алгоритм-классификатор. Такой алгоритм называют линейным классификатором. Этот алгоритм должен брать какое-то число объектов и относить их к одной из групп, или, как говорят в машинном обучении, к одному из классов. На выходе мы должны получить метку класса, к которому относится объект. Первая метка — -1 (минус единица) — человек не выплатит кредит в срок, вторая метка 1 (единица) — человек выплатит все в срок. Чуть позже мы разберем, почему взяли именно такие метки классов.

Какие инструменты для решения задачи у нас есть на данный момент? У нас есть алгоритм линейной регрессии. Напомню, что цель линейной регрессии — найти такую прямую на графике, которая наиболее точно будет отображать зависимость целевой переменной от признаков объектов. То есть, нам нужно как-то ввести какое-то условие, чтобы прямая разделяла объекты.

Итак, посмотрим на формулу линейной регрессии. Обратите внимание на вид формулы: свободный коэффициент здесь для всех признаков равен единице. Поэтому мы и можем записать формулу в таком виде.



Линейная регрессия

Пусть w_0 будет признаком, который для всех элементов будет равен единице. Тогда наша формула приобретет вид:

$$a(x) = \langle wx \rangle$$

Раз нам нужно определить всего два класса, то мы можем просто сравнивать скалярное произведение с нулем! Если он будет больше нуля — то объект будет относиться к положительному классу, а если меньше нуля — к отрицательному!



Бинарный классификатор

sign - знак скалярного произведения


$$a(x) = \text{sign} \langle wx \rangle$$

Отлично, у нас получилась формула бинарного классификатора! Обратите внимание, что у нас получился именно линейный бинарный классификатор, так как

по сути мы получили с вами уравнение гиперплоскости. А в двумерном пространстве уравнение гиперплоскости есть прямая, которая как раз и записывается таким уравнением.

Вы также можете увидеть другой вид записи бинарного классификатора:

Линейный классификатор



Бинарный классификатор

[] - индикатор события

$$a(x) = [(w, x) > 0]$$

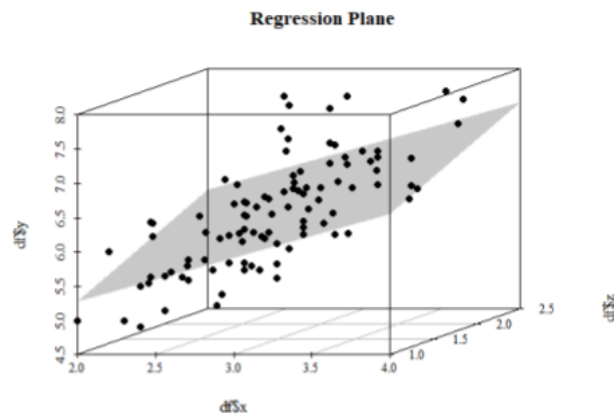
В такой записи квадратными скобками обозначается так называемый индикатор. Формула читается так: **предсказание модели a от объектов x это индикатор того события, что скалярное произведение весов и признаков объекта больше нуля.**

Стоит добавить еще одно очень важное определение. В данном примере у нас есть всего лишь один признак объекта. На прошлом занятии в линейной регрессии у нас также был один признак. А что же делать, если признаков много? Ведь в реальной задаче их может быть и 10, и 20, и 50.

В таком случае тоже будет гиперплоскость, просто более высокой размерности. Например, для регрессии она может выглядеть так:



Гиперплоскость



Аналогично она может выглядеть и для классификации

Ну а теперь давайте сформируем решение по нашей задаче. Возьмем несколько признаков для примера. Пусть x_1 — зарплата клиента банка, x_2 — количество трат в месяц, x_3 — задолженность по прошлым кредитам. Тогда наше уравнение примет такой вид как на слайде:



Решение задачи

$$a(x) = \text{sign}(w_0 + w_1x_1 + w_2x_2 + w_3x_3)$$

x_1 - зарплата клиента банка

x_2 - количество трат в месяц

x_3 - задолженность по прошлым кредитам

Веса мы найдем в процессе обучения. Если обобщить — линейный классификатор — простой алгоритм, который является основой логистической регрессии, про которую мы поговорим в следующем блоке. В таком виде данный классификатор почти не используется, но когда мы будем говорить про ансамбли, мы обязательно про него вспомним.

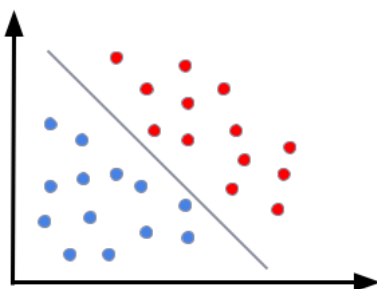
Ну что же, задача решена!

Логистическая регрессия

Теперь давайте подумаем, как можно лучше отразить классы в нашей задаче. Давайте сделаем так, чтобы наш классификатор мог предсказывать не только класс объекта, но и вероятность такого класса. Согласитесь, если человек вернет кредит с 90 процентной вероятностью — это будет здорово. Но если человек вернет кредит с 60 процентной вероятностью — это уже не очень хорошо. Хотя класс не будет отличаться, и в первом и во втором случае классификатор выдаст единицу. Поэтому давайте найдем способ, как говорят в машинном обучении, мягко классифицировать объекты, то есть узнать вероятность класса.

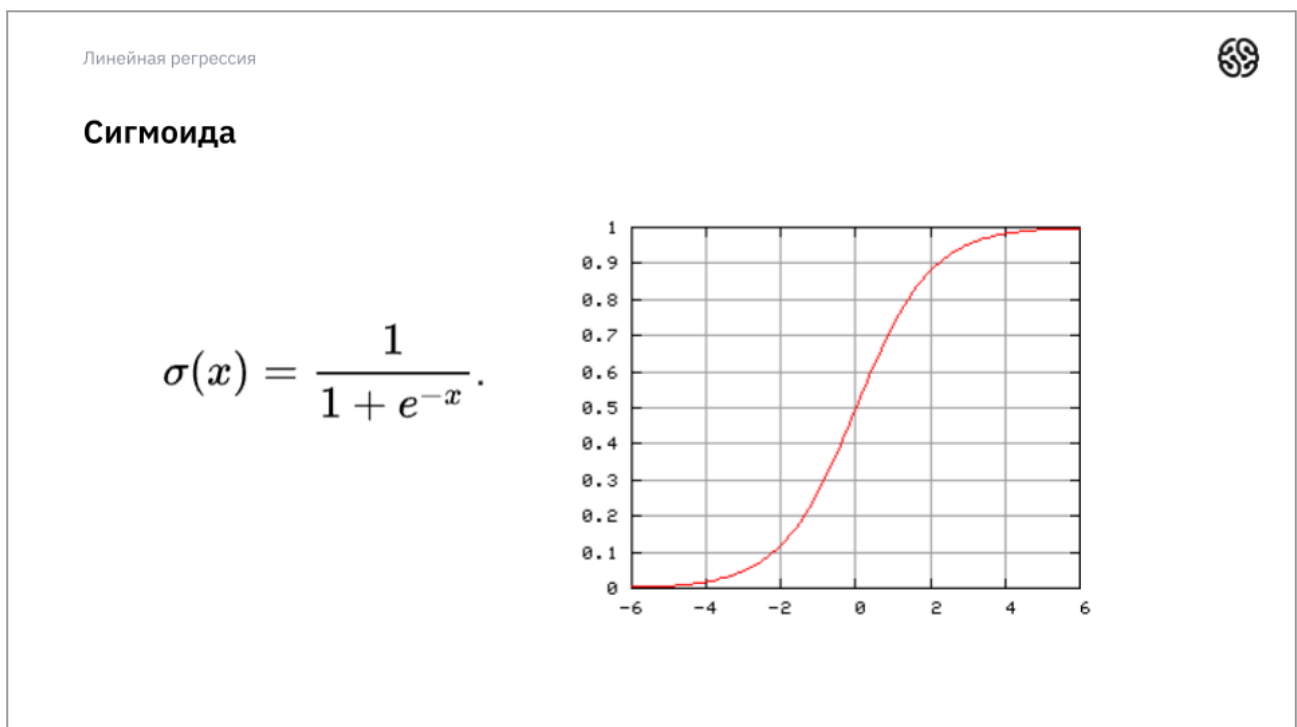
Если нам нужно узнать вероятность, наш линейный классификатор приобретет другое название — логистическая регрессия. Это частный случай линейного классификатора. И сейчас мы разберемся как это работает.

Уверенность классификатора



Посмотрите на слайд — это пример нашего линейного классификатора. Обратите внимание — часть точек находится близко от разделяющей плоскости, а часть — достаточно далеко. Чем дальше находятся объекты от разделяющей плоскости, тем больше уверенность классификатора в предсказании на этих объектах. Но это расстояние может быть произвольным числом — как близким к нулю, так и очень большим. А нам нужно получить вероятность — то есть число от нуля до единицы. В этом и заключается сейчас наша задача.

Для того чтобы перевести расстояние от объекта до разделяющей прямой в отрезок от нуля до единицы, мы воспользуемся специальной функцией, которая называется сигмоида:



Сигмоида переводит числа в отрезок от нуля до единицы, а это как раз то, что нам нужно. Давайте добавим формулу сигмоиды вместо знака к нашей формуле линейного классификатора. Тогда у нас получится следующая формула:



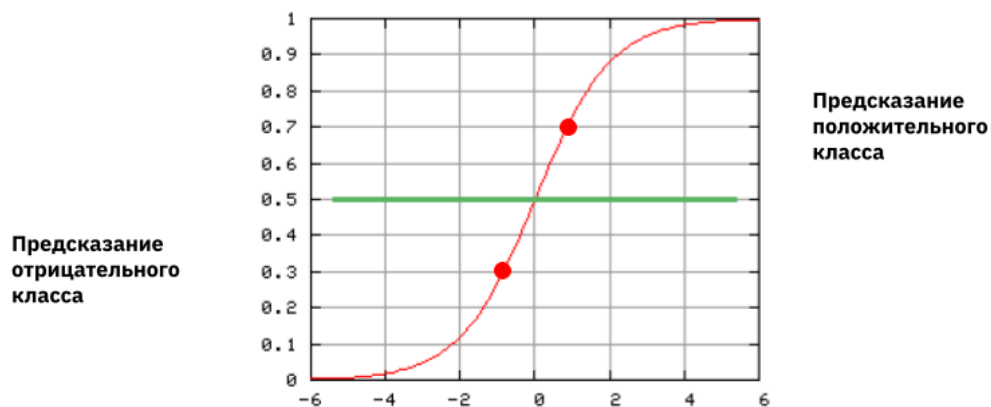
Формула логистической регрессии

$$a(x) = \frac{1}{1 + e^{-(w,x)}}$$

Сигмоида относит все что выше 0.5 к положительному классу, а все что ниже к отрицательному.



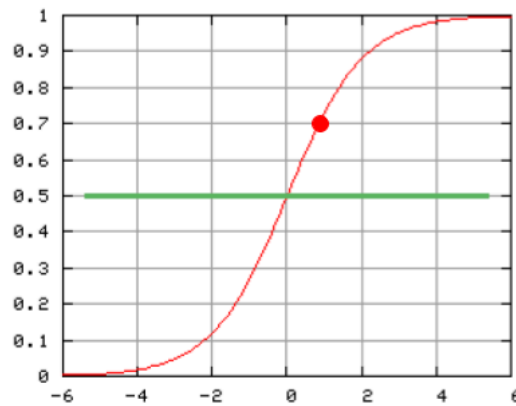
Сигмоида



Обратите внимание — чем увереннее предсказание, тем дальше оно находится от разделяющей границы. В предсказаниях на слайде классификатор уверен, поэтому расположил их достаточно далеко:



Сигмоида

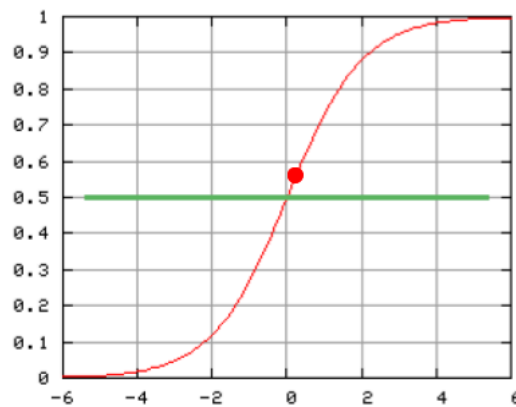


Предсказание
положительного
класса

А вот если бы предсказание было с небольшой вероятностью, мы бы увидели нечто подобное.



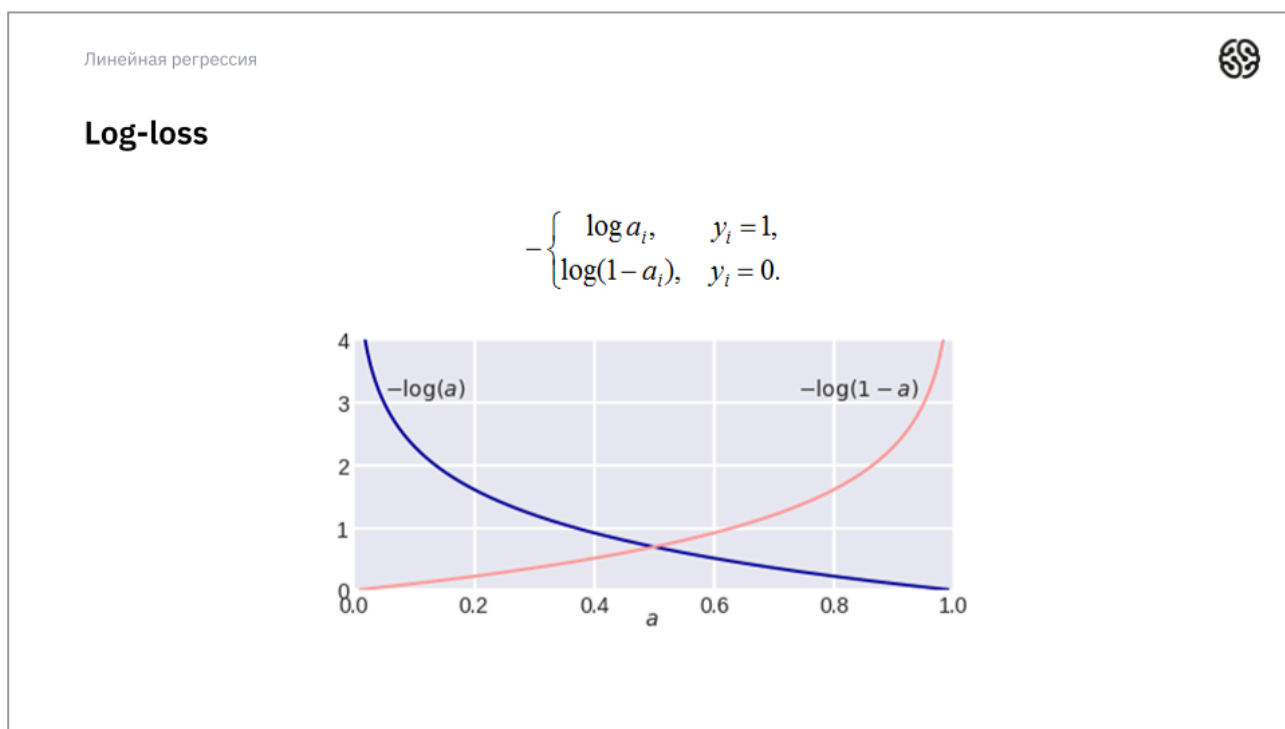
Сигмоида



Предсказание
положительного
класса с
вероятностью чуть
больше 0.5

Давайте суммируем все вышесказанное: в результате классификации у нас получается некое число, то есть класс объекта. Уверенность классификатора по поводу предсказания этого класса — это расстояние до разделяющей прямой. Чем дальше находится объект от разделяющей прямой — тем увереннее это предсказание. Это расстояние — произвольная величина. Чтобы отразить уверенность классификатора по поводу предсказания мы используем функцию, которая называется сигмоида. Эта функция способна переводить числа в отрезок от нуля до единицы, тем самым отражая вероятность этого класса. А зная вероятность можно оценить, насколько классификатор уверен в предсказании. Такой классификатор, который может предсказать не только класс, но и его вероятность, называется **логистическая регрессия**. Логистическая регрессия, так же как и линейная регрессия — легко интерпретируемая модель. Глядя на формулу и подставив значения признаков и весов можно с легкостью сказать, какой признак влияет больше на попадание в класс.

Едем дальше! Формулу для предсказаний мы получили. Теперь давайте разберемся, а какую же функцию мы будем минимизировать. А минимизировать ее мы будем с помощью функции log-loss:



Ух, выглядит непросто! Давайте разберемся как это работает. Логарифмическая потеря указывает, насколько близка вероятность предсказания к соответствующему истинному значению (0 или 1 в случае двоичной классификации). Чем больше прогнозируемая вероятность отклоняется от фактического значения, тем выше значение логарифма потерь.

К сожалению, вывод такой формулы остается за рамками нашего курса, так как он использует методы математической статистики и в частности методы правдоподобия. Вы можете вспомнить математику из прошлых курсов и самостоятельно вывести эту формулу! На самом деле линейная регрессия и логистическая регрессия имеют определенное родство, так как функции их потерь основаны на нормальном распределении и на распределении Бернулли.

Давайте посмотрим как ведет себя log-loss на одном объекте. В зависимости от того, к какому классу относится объект, мы считаем либо минус логарифм от предсказания модели, либо минус логарифм от единицы минус предсказание модели.

Стоит обратить внимание на важную деталь: минимизация функции log-loss не просто дает число из отрезка от нуля до единицы, log-loss обеспечивает предсказание именно математической вероятности.



Модель логистической регрессии

Формула логистической регрессии:

$$a(x) = \frac{1}{1 + e^{-(w,x)}}$$

Функция потерь, которую мы минимизируем:

$$H(w) = -\frac{1}{N} \sum_{i=1}^N y_i \times \log(p(y_i)) + (1 - y_i) \times \log(1 - p(y_i))$$

Итак, нам осталось разобраться только с уменьшением функции потерь. Если в линейной регрессии это метод наименьших квадратов, то в логистической регрессии нет явной формулы для уменьшения функции потерь. Поэтому пришло время познакомиться с мощным алгоритмом оптимизации, который применяется не только в линейных моделях, таких как линейная и логистическая регрессия, но и в нейросетях.

Такой алгоритм называется градиентный спуск. Его можно применять в любых дифференцируемых функциях.

💡 Дифференцируемая функция — функция, которая может быть хорошо приближена линейной функцией. Говоря простым языком — если мы можем взять некую функцию $a(x)$, и описать эту функцию с помощью прямой, то функция $a(x)$ является дифференцируемой функцией.

Градиентный спуск

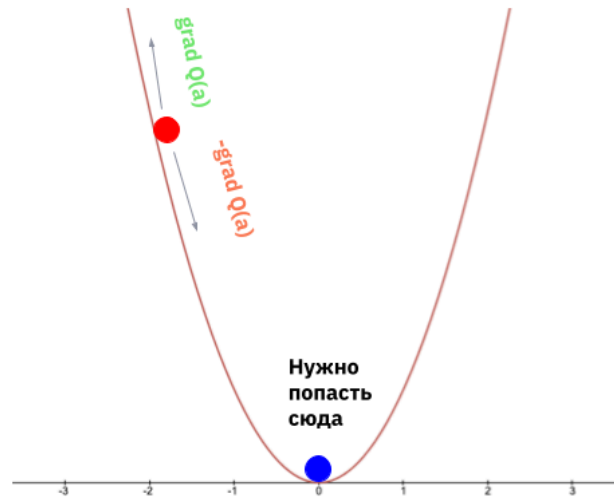
Давайте разберемся на примере.

Нам нужно найти минимум функции, которая изображена на слайде. Давайте предположим, что мы находимся в процессе обучения модели и находимся в некоторой точке на графике. Сразу возникает несколько вопросов: как понять в какую сторону двигаться и с каким шагом?



Здесь нужно вспомнить математику. Что нам может показать направление? Мы знаем, что производная, или в общем случае градиент, показывает нам направление возрастания функции. Следовательно, нам нужно двигаться в сторону, противоположную градиенту:

Как найти минимум функции?

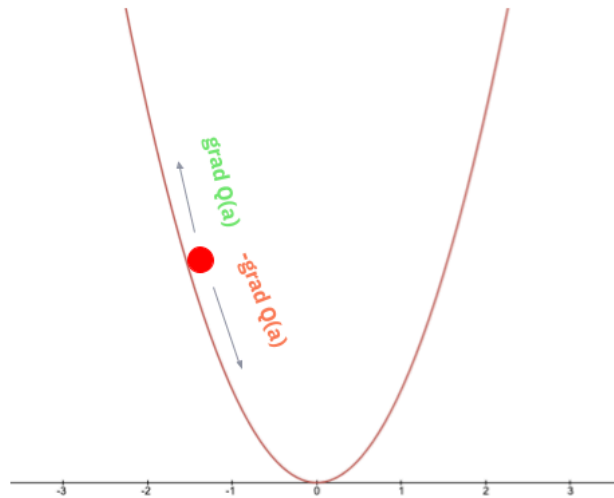


Производная характеризует скорость изменения функции. Для того чтобы в полной мере понять это определение, обратитесь к дополнительным материалам в конце конспекта, там вы найдете ресурсы, на которых подробно объясняется этот термин.

Соответственно, на каждом шаге обучения модели, мы вычисляем градиент и двигаемся в противоположную сторону. У нас получается итеративный процесс, в котором мы повторяем одни и те же действие какое-то количество раз.



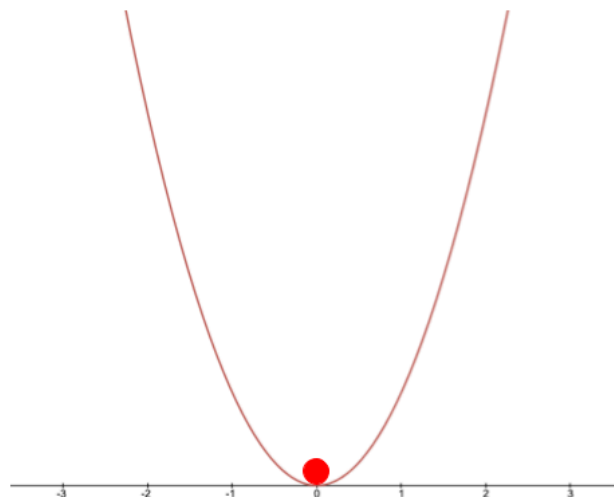
Как найти минимум функции?



Конечно же, рано или поздно, мы приблизимся к минимуму функции.



Как найти минимум функции?



Это и называется градиентным спуском. Формулу вы можете увидеть на слайде.

Градиентный спуск



Градиентный спуск

Вычисляем градиент $Q(w)$ в точке и двигаемся в противоположную сторону

$$W_{new} = W_{old} - Q'(W_{old})$$

Важным параметром градиентного спуска является градиентный шаг. Градиентный шаг — это гиперпараметр, который мы задаем вручную. Он регулирует обновление весов градиентного спуска. Градиентный шаг часто называют ещё скоростью обучения. По факту так и получается: чем меньше шаг, тем дольше будет обучаться наша модель. Это работает и наоборот. Поэтому градиентный шаг подбирают в зависимости от того, как проходит обучение модели. Обозначим градиентный шаг буквой альфа. Добавим его в формулу:



Градиентный спуск

Добавляем гипер параметр - шаг градиентного спуска:

$$W_{new} = W_{old} - \alpha \times Q'(W_{old})$$

Мы имеем дело с производной только в одномерном случае. По факту, почти все функции, которые мы минимизируем — это функции нескольких аргументов. Ну а как вы уже знаете, число аргументов это по сути число весов. Поэтому давайте будем учитывать это в нашей формуле, которая примет векторный вид, а вместо производной мы будем считать градиент — многомерное обобщение производной.



Градиентный спуск

W - вектор, тогда:

$$W_{new} = W_{old} - \alpha \times \nabla Q(W_{old})$$

В свою очередь градиент — это вектор состоящий из производных функций по всем своим аргументам.

Перевернутый треугольник называется набла. Набла Q — это градиент функции потерь.

Этот метод очень общий, поэтому он применяется для обучения практически всех алгоритмов машинного обучения, то есть для минимизации почти всех функций потерь.

У градиентного спуска есть и ряд недостатков. На каждой итерации метода мы вычисляем градиент функции потерь, то есть производную функции потерь по всем параметрам, а так же мы вычисляем этот градиент на всех объектах обучающей выборки. Это довольно долгий процесс, причем нам нужно хранить все промежуточные значения в памяти компьютера. Но можно сэкономить ресурс.

Разберемся как это сделать. Давайте не считать градиент на всех объектах, а выбирать случайный объект и считать градиент только для него. Таким образом мы все равно придем к минимуму функции

В этом случае шаг градиентного спуска выглядит следующим образом. Сначала мы выбираем один случайный объект из обучающей выборки, а затем мы обновляем веса по известной формуле, вычисляя градиент только по этому объекту. Такая модификация метода называется стохастический градиентный спуск.

Градиентный спуск



Стохастический градиентный спуск

Считаем градиент не по всем объектам, а выбираем случайный объект.

Формула останется такой же, но под второй частью выражения мы будем считать только один элемент.

$$W_{new} = W_{old} - \alpha \times \nabla Q(W_{old})$$

В классическом градиентном спуске мы четко двигаемся к минимуму с каждым шагом потому что мы двигаемся строго по антиградиенту. СГП приблизительно вычисляет минимум на каждой итерации, поэтому наше движение к минимуму происходит менее четко и менее строго, но в среднем за много шагов мы тоже приходим к минимуму.



Однако стохастический градиентный спуск, как и классический градиентный спуск, рано или поздно придет к минимуму функции потерь, но при этом сильно сэкономит нам вычислительные ресурсы.

Именно стохастический градиентный спуск чаще всего используют на практике, так как он более экономный по вычислительным ресурсам.

А теперь давайте перейдем к практике!

Практика

Наша практика будет состоять из трех частей:

- в первой части мы реализуем алгоритм градиентного спуска на примере несложной функции
- во второй части мы применим градиентный спуск в линейной регрессии
- в третьей части мы обучим новую модель с помощью логистической регрессии

Градиентный спуск на практике

```
# установим размер графика
fig = plt.figure(figsize = (12,10))

# создадим последовательность из 1000 точек в интервале от -5 до 5
# для осей w1 и w2
w1 = np.linspace(-5, 5, 1000)
w2 = np.linspace(-5, 5, 1000)

# создадим координатную плоскость из осей w1 и w2
w1, w2 = np.meshgrid(w1, w2)

# пропишем функцию
f = w1 ** 2 + w2 ** 2

# создадим трехмерное пространство
ax = fig.add_subplot(projection = '3d')

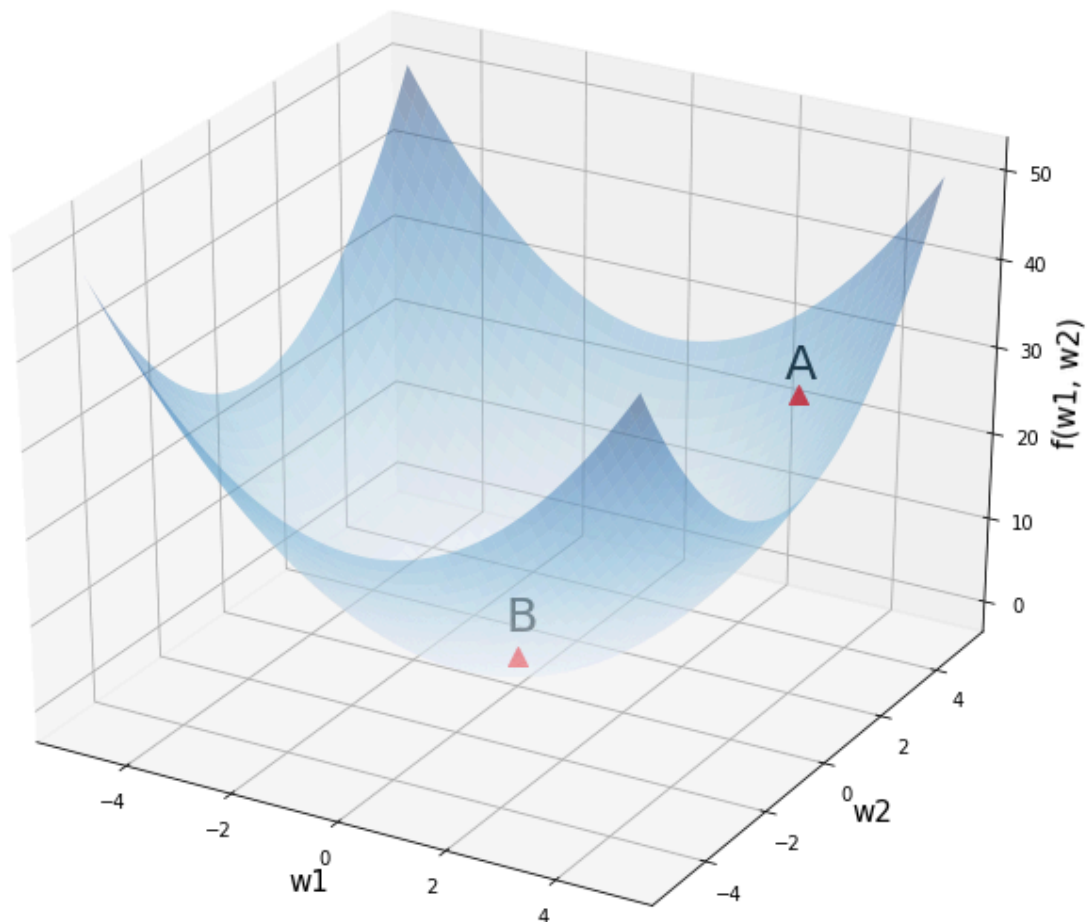
# выведем график функции, alpha задает прозрачность
ax.plot_surface(w1, w2, f, alpha = 0.4, cmap = 'Blues')

# выведем точку A с координатами (3, 4, 25) и подпись к ней
ax.scatter(3, 4, 25, c = 'red', marker = '^', s = 100)
ax.text(3, 3.5, 28, 'A', size = 25)

# аналогично выведем точку B с координатами (0, 0, 0)
ax.scatter(0, 0, 0, c = 'red', marker = '^', s = 100)
ax.text(0, -0.4, 4, 'B', size = 25)
```

```
# укажем подписи к осям
ax.set_xlabel('w1', fontsize = 15)
ax.set_ylabel('w2', fontsize = 15)
ax.set_zlabel('f(w1, w2)', fontsize = 15)
```

```
# выведем результат
plt.show()
```



```
# установим размер графика
fig, ax = plt.subplots(figsize = (10,10))

# создадим последовательность из 100 точек в интервале от -5 до 5
# для осей w1 и w2
w1 = np.linspace(-5.0, 5.0, 100)
w2 = np.linspace(-5.0, 5.0, 100)

# создадим координатную плоскость из осей w1 и w2
w1, w2 = np.meshgrid(w1, w2)
```

```

# пропишем функцию
C = w1 ** 2 + w2 ** 2

# построим изолинии (линии уровня)
plt.contourf(w1, w2, C, cmap = 'Blues')

# выведем точку A с координатами на плоскости (3, 4)
ax.scatter(3, 4, c = 'red', marker = '^', s = 200)
ax.text(2.85, 4.3, 'A', size = 25)

# и точку B с координатами (0, 0)
ax.scatter(0, 0, c = 'red', marker = '^', s = 200)
ax.text(-0.15, 0.3, 'B', size = 25)

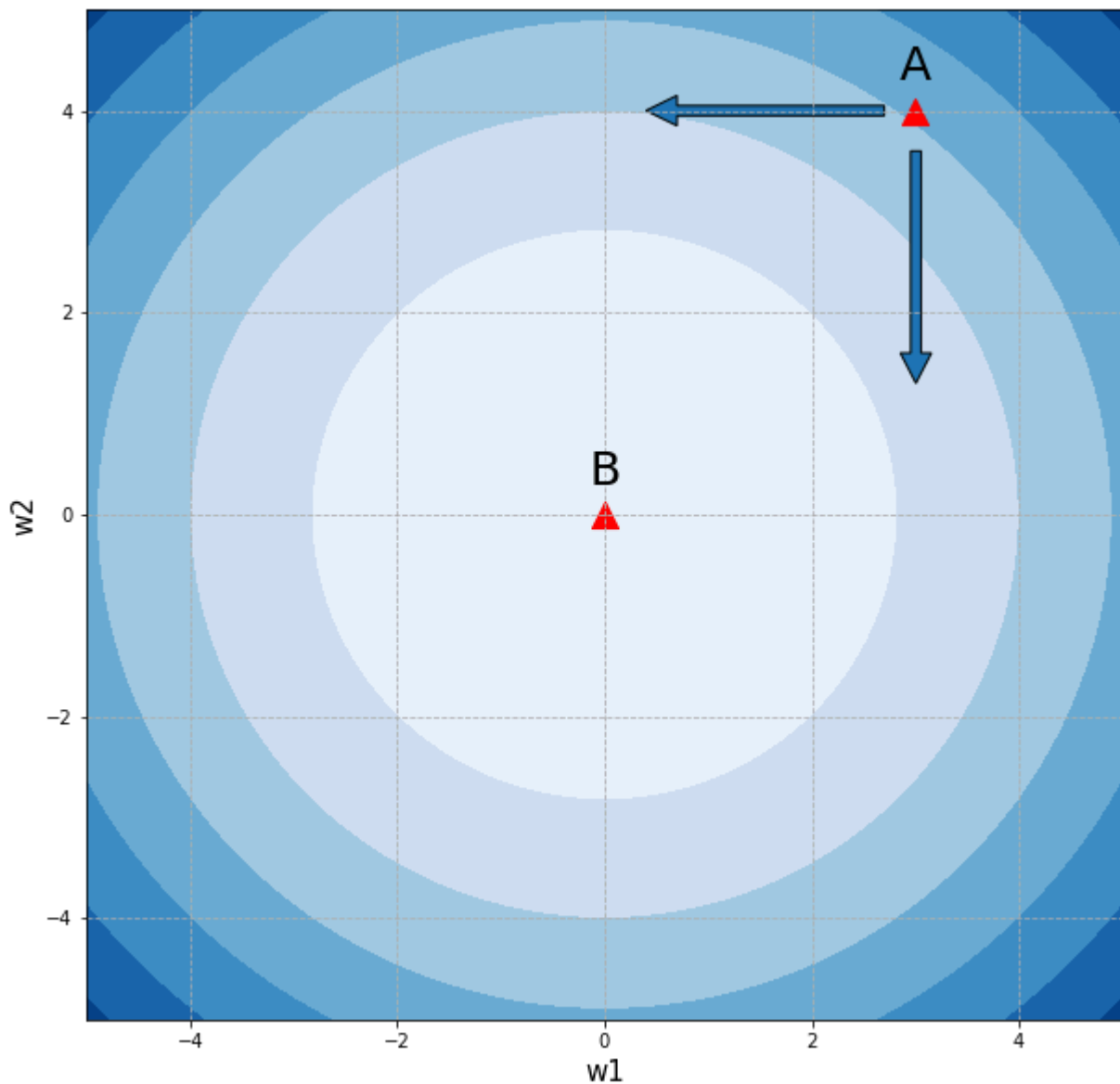
# укажем подписи к осям
ax.set_xlabel('w1', fontsize = 15)
ax.set_ylabel('w2', fontsize = 15)

# а также стрелки направления изменений вдоль w1 и w2
ax.arrow(2.7, 4, -2, 0, width = 0.1, head_length = 0.3)
ax.arrow(3.005, 3.6, 0, -2, width = 0.1, head_length = 0.3)

# создадим сетку в виде прерывистой черты
plt.grid(linestyle = '--')

# выведем результат
plt.show()

```

```
# установим размер графика
fig, ax = plt.subplots(figsize = (10,10))

# создадим последовательность из 100 точек в интервале от -10 до 10
# для осей w1 и w2
w1 = np.linspace(-10.0, 10.0, 100)
w2 = np.linspace(-10.0, 10.0, 100)

# создадим координатную плоскость из осей w1 и w2
w1, w2 = np.meshgrid(w1, w2)

# пропишем функцию
C = w1 ** 2 + w2 ** 2
```

```

# построим изолинии (линии уровня)
plt.contourf(w1, w2, C, cmap = 'Blues')

# выведем точку A с координатами на плоскости (3, 4)
ax.scatter(3, 4, c = 'red', marker = '^', s = 200)
ax.text(2.5, 4.5, 'A', size = 25)

# и точку B с координатами (0, 0)
ax.scatter(0, 0, c = 'red', marker = '^', s = 200)
ax.text(-1, 0.3, 'B', size = 25)

# укажем подписи к осям
ax.set_xlabel('w1', fontsize = 15)
ax.set_ylabel('w2', fontsize = 15)

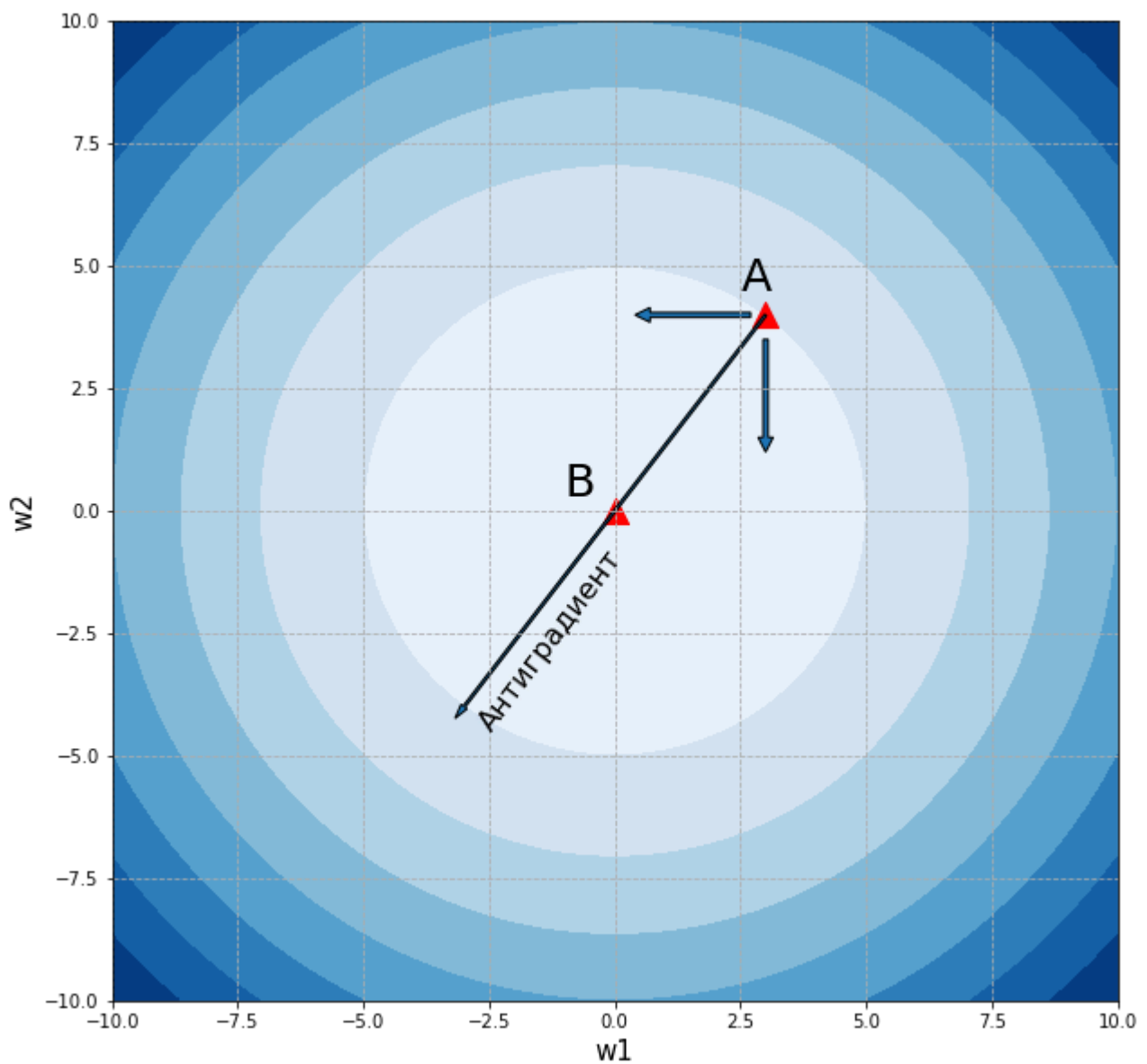
# а также стрелки направления изменений вдоль w1 и w2
ax.arrow(2.7, 4, -2, 0, width = 0.1, head_length = 0.3)
ax.arrow(3, 3.5, 0, -2, width = 0.1, head_length = 0.3)

# выведем вектор антиградиента с направлением (-6, -8)
ax.arrow(3, 4, -6, -8, width = 0.05, head_length = 0.3)
ax.text(-2.8, -4.5, 'Антиградиент', rotation = 53, size = 16)

# создадим сетку в виде прерывистой черты
plt.grid(linestyle = '--')

# выведем результат
plt.show()

```



```
# пропишем функцию потерь
def objective(w1, w2):
    return w1 ** 2 + w2 ** 2

# а также производную по первой
def partial_1(w1):
    return 2.0 * w1

# и второй переменной
def partial_2(w2):
    return 2.0 * w2

# пропишем изначальные веса
w1, w2 = 3, 4

# количество итераций
```

```

iter = 100

# и скорость обучения
learning_rate = 0.05
# создадим списки для учета весов и уровня ошибки
w1_list, w2_list, l_list = [], [], []
# в цикле с заданным количеством итераций
for i in range(iter):

    # будем добавлять текущие веса в соответствующие списки
    w1_list.append(w1)
    w2_list.append(w2)

    # и рассчитывать и добавлять в список текущий уровень ошибки
    l_list.append(objective(w1, w2))

    # также рассчитаем значение частных производных при текущих весах
    par_1 = partial_1(w1)
    par_2 = partial_2(w2)

    # будем обновлять веса в направлении,
    # обратном направлению градиента, умноженному на скорость обучения
    w1 = w1 - learning_rate * par_1
    w2 = w2 - learning_rate * par_2

# выведем итоговые веса модели и значение функции потерь
w1, w2, objective(w1, w2)
fig = plt.figure(figsize = (14,12))

w1 = np.linspace(-5, 5, 1000)
w2 = np.linspace(-5, 5, 1000)

w1, w2 = np.meshgrid(w1, w2)

f = w1 ** 2 + w2 ** 2

ax = fig.add_subplot(projection = '3d')

ax.plot_surface(w1, w2, f, alpha = 0.4, cmap = 'Blues')

ax.text(3, 3.5, 28, 'A', size = 25)
ax.text(0, -0.4, 4, 'B', size = 25)

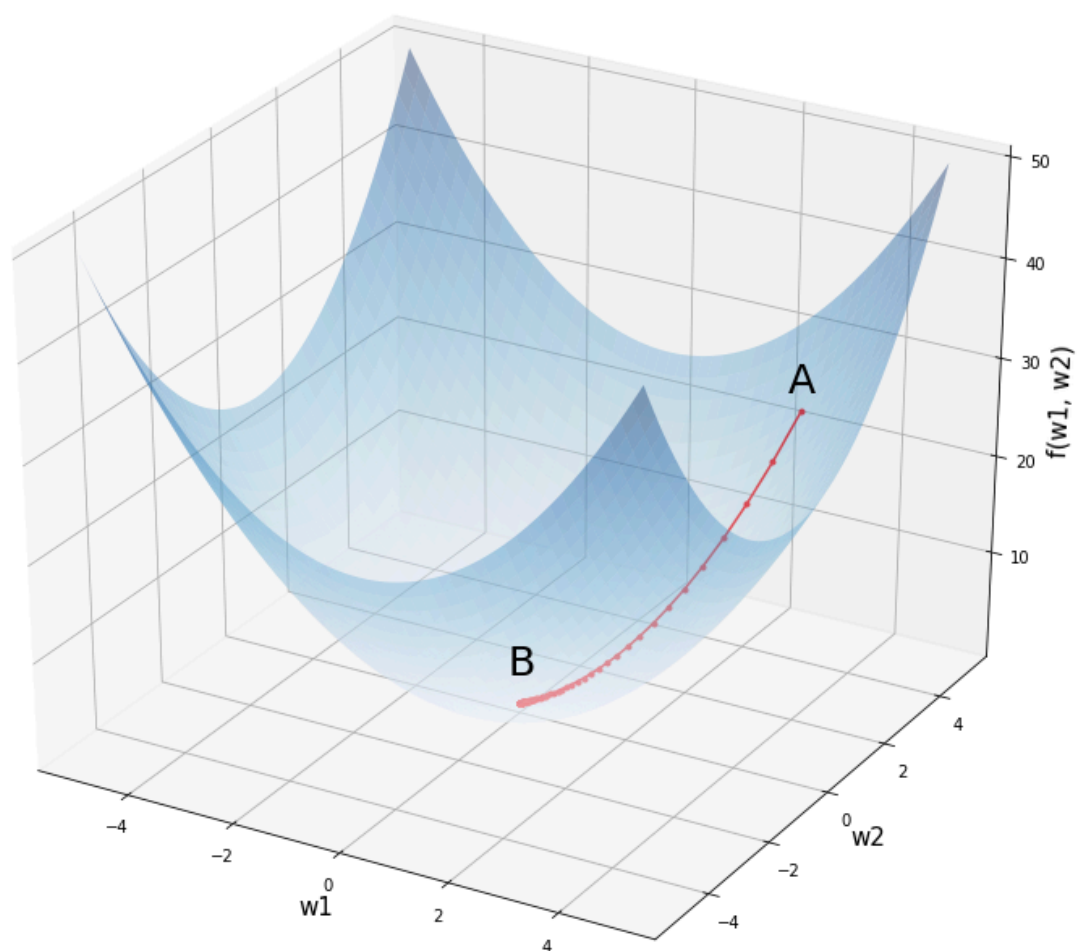
ax.set_xlabel('w1', fontsize = 15)
ax.set_ylabel('w2', fontsize = 15)

```

```
ax.set_zlabel('f(w1, w2)', fontsize = 15)

# выведем путь алгоритма оптимизации
ax.plot(w1_list, w2_list, l_list, '.-', c = 'red')

plt.show()
```



Работа градиентного спуска на примере линейной регрессии

В этой практике мы воспользуемся данными из одного датасета, который описывает зависимость роста от обхвата шеи у женщин. Мы построим алгоритм линейной регрессии и воспользуемся градиентным спуском для уменьшения функции потерь! Обратите внимание, что данные действия мы будем делать вручную, без использования библиотеки Sklearn с готовыми моделями.

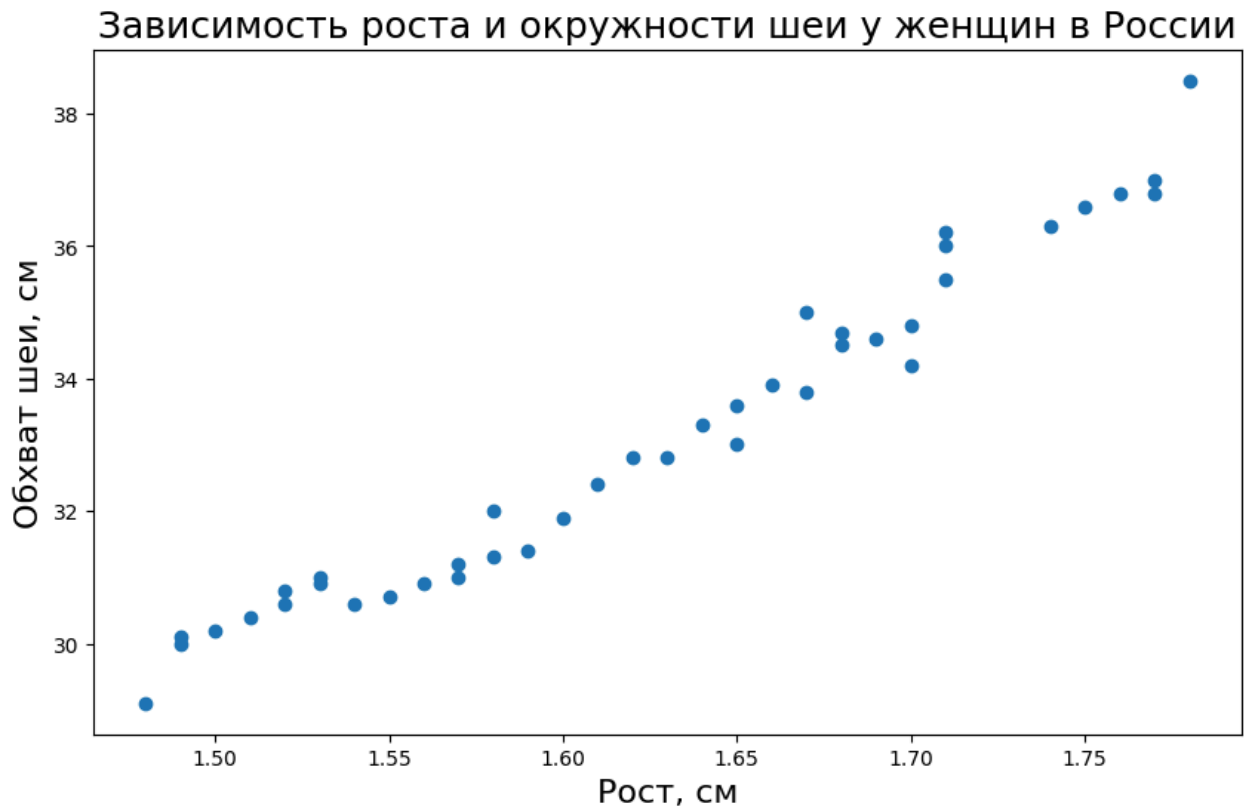
```
# Импорт библиотеки numpy для работы с массивами и matplotlib для
визуализации данных
import numpy as np
import matplotlib.pyplot as plt

# Задаем массивы данных роста (X) и обхвата шеи (y)
X = np.array([1.48, 1.49, 1.49, 1.50, 1.51, 1.52, 1.52, 1.53, 1.53,
1.54, 1.55, 1.56, 1.57, 1.57, 1.58, 1.58, 1.59, 1.60, 1.61, 1.62, 1.63,
1.64, 1.65, 1.65, 1.66, 1.67, 1.67, 1.68, 1.68, 1.69, 1.70, 1.70,
1.71, 1.71, 1.71, 1.74, 1.75, 1.76, 1.77, 1.77, 1.78])
y = np.array([29.1, 30.0, 30.1, 30.2, 30.4, 30.6, 30.8, 30.9, 31.0,
30.6, 30.7, 30.9, 31.0, 31.2, 31.3, 32.0, 31.4, 31.9, 32.4, 32.8, 32.8,
33.3, 33.6, 33.0, 33.9, 33.8, 35.0, 34.5, 34.7, 34.6, 34.2, 34.8, 35.5,
36.0, 36.2, 36.3, 36.6, 36.8, 36.8, 37.0, 38.5])

# Создаем точечную диаграмму для визуализации данных
plt.figure(figsize = (10,6))
plt.scatter(X, y)

# Добавляем подписи к осям и заголовок к графику
plt.xlabel('Рост, см', fontsize = 16)
plt.ylabel('Обхват шеи, см', fontsize = 16)
plt.title('Зависимость роста и окружности шеи у женщин в России',
fontsize = 18)

# Отображаем график
plt.show()
```



Определение функции линейной регрессии

```
def regression(X, w, b):
    return w * X + b
```

Определение функции потерь

```
def objective(X, y, w, b, n):
    return np.sum((y - regression(X, w, b)) ** 2) / (2 * n)
```

Определение частной производной функции потерь по весу w

```
def partial_w(X, y, w, b, n):
    return np.sum(-X * (y - (w * X + b))) / n
```

Определение частной производной функции потерь по смещению b

```
def partial_b(X, y, w, b, n):
    return np.sum(-(y - (w * X + b))) / n
```

Функция для выполнения градиентного спуска

```
def gradient_descent(X, y, iter, learning_rate):
    w, b = 0, 0 # Начальные значения параметров модели
    n = len(X) # Количество точек данных
```

Списки для хранения истории значений параметров и функции потерь

```
w_list, b_list, l_list = [], [], []
```

```

# Основной цикл градиентного спуска
for i in range(iter):
    w_list.append(w)
    b_list.append(b)
    l_list.append(objective(X, y, w, b, n))

# Вычисление градиентов
par_1 = partial_w(X, y, w, b, n)
par_2 = partial_b(X, y, w, b, n)

# Обновление параметров
w = w - learning_rate * par_1
b = b - learning_rate * par_2

return w_list, b_list, l_list

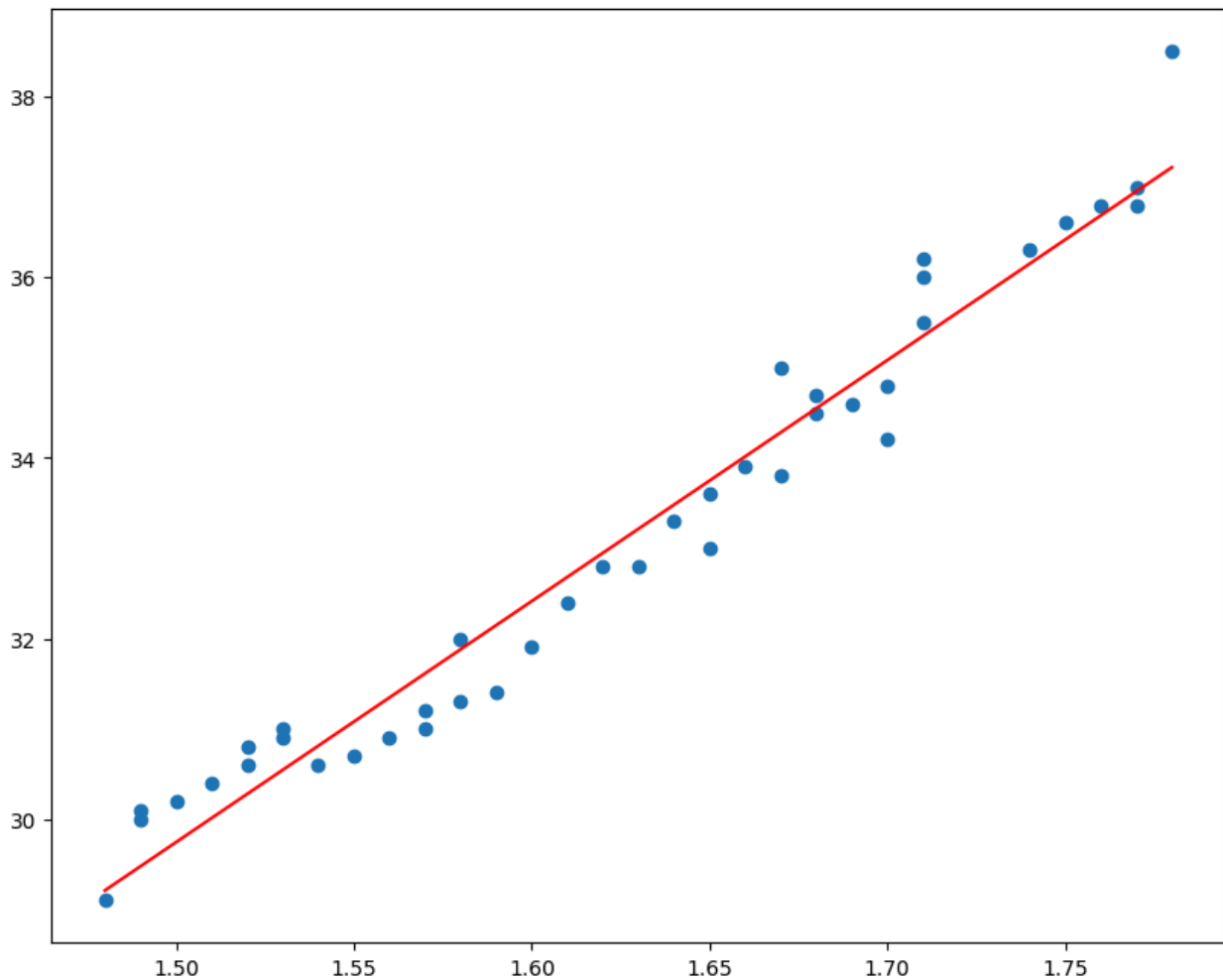
# Выполнение градиентного спуска
w_list, b_list, l_list = gradient_descent(X, y, iter = 200000,
learning_rate = 0.01)

# Выводим последние значения веса, смещения и функции потерь
print(w_list[-1], b_list[-1], l_list[-1])

# Получаем предсказания модели
y_pred_gd = regression(X, w_list[-1], b_list[-1])

# Создаем новый график для визуализации результатов регрессии
plt.figure(figsize = (10, 8))
plt.scatter(X, y) # Точечная диаграмма исходных данных
plt.plot(X, y_pred_gd, 'r') # Линия предсказаний модели
plt.show()

```

Логистическая регрессия на практике

В этой практике мы создадим модель логистической регрессии для классификации ирисов. Мы воспользуемся набором данных ирисы Фишера - это очень популярный датасет, который описывает четыре типа ирисов по четырем характеристикам.

Еще один интересный нюанс - мы будем использовать модуль SKlearn. Это универсальный модуль, который содержит уже готовые классы практически всех алгоритмов классического машинного обучения.

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
```

```
# Загрузка набора данных Iris
```

```
iris = load_iris()

# Извлечение первых двух признаков
x = iris.data[:, :2]
y = iris.target

# Инициализация классификатора логистической регрессии
clf = LogisticRegression()

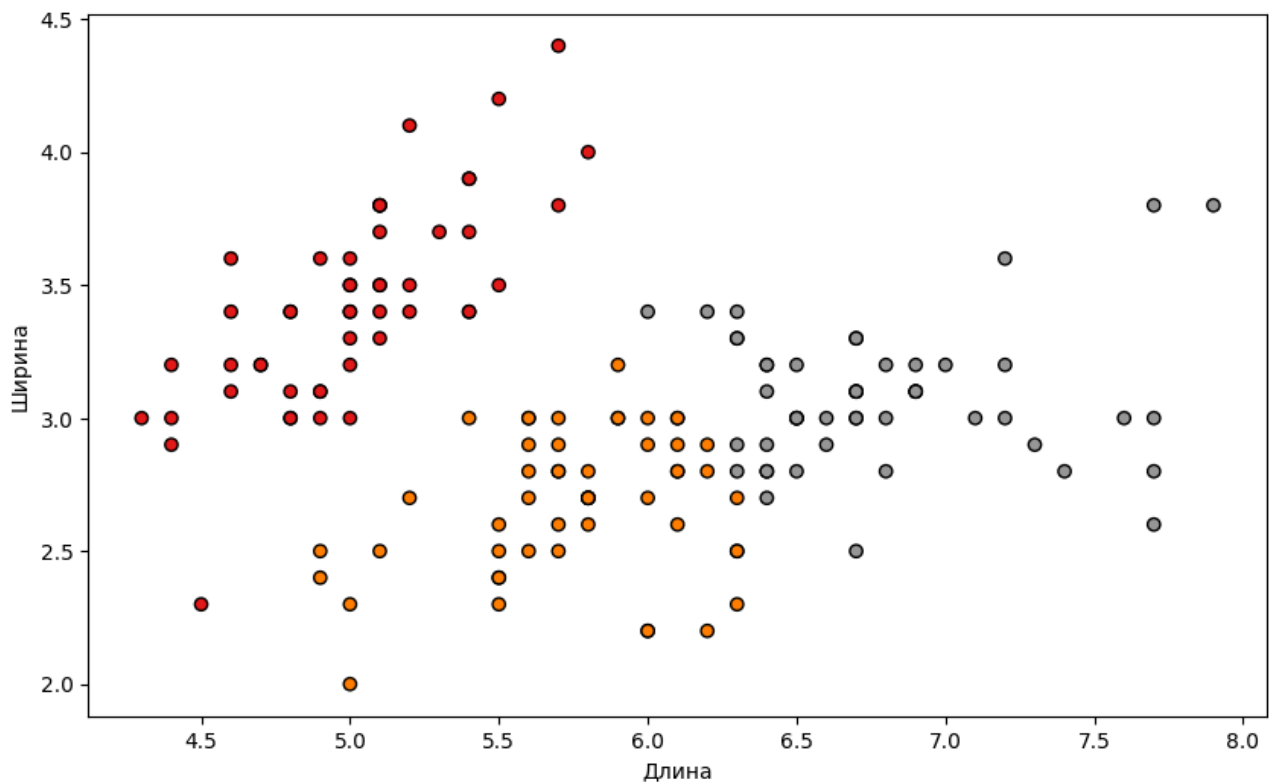
# Обучение классификатора на данных
clf.fit(x, y)

# Прогнозирование меток классов для данных
y_pred = clf.predict(x)

# Построение графика точек данных с предсказанными метками
plt.figure(figsize=(10, 6))
plt.scatter(x[:, 0], x[:, 1], c=y_pred, cmap=plt.cm.Set1, edgecolor='k')

# Задание подписей для осей x и y
plt.xlabel('Длина')
plt.ylabel('Ширина')

# Отображение графика
plt.show()
```



Вся практика по ссылке:

<https://colab.research.google.com/drive/1amMG7cMw5PudZkXR6mQChzmbqV27GU26?usp=sharing>

Выводы

Логистическая регрессия — мощный алгоритм для классификации объектов, который позволяет узнать не только принадлежность объекта к классу, но и вычислить вероятность такой принадлежности. Однако иногда бывает полезен и простой линейный классификатор. Все зависит от сложности задачи. Чуть позже, когда мы познакомимся с композициями, вы узнаете, что линейный классификатор используется как алгоритм, дополняющий другие несложные алгоритмы.

Градиентный спуск — один из самых мощных способов поиска минимума функции, который используется также при обучении нейросетей.

Что можно почитать еще?

1. <https://education.yandex.ru/handbook/data-analysis/article/logisticheskaya-regressiya> - логистическая регрессия и градиентный спуск
2. <https://habr.com/ru/companies/skillfactory/articles/714244/> - подробнее о разных видах логистической регрессии

Используемая литература

1. Бринк Х., Ричардс Дж., Феверолф М. - Машинное обучение, 2017 год
2. Андрей Бурков — Машинное обучение без лишних слов, 2020 год
3. Андреас Мюллер, Сара Гвидо — Введение в машинное обучение с помощью Python, 2017 год
4. Джереми Уатт, Реза Борхани, Аггелос Катсаггелос — Машинное обучение: основы, алгоритмы и практика применения