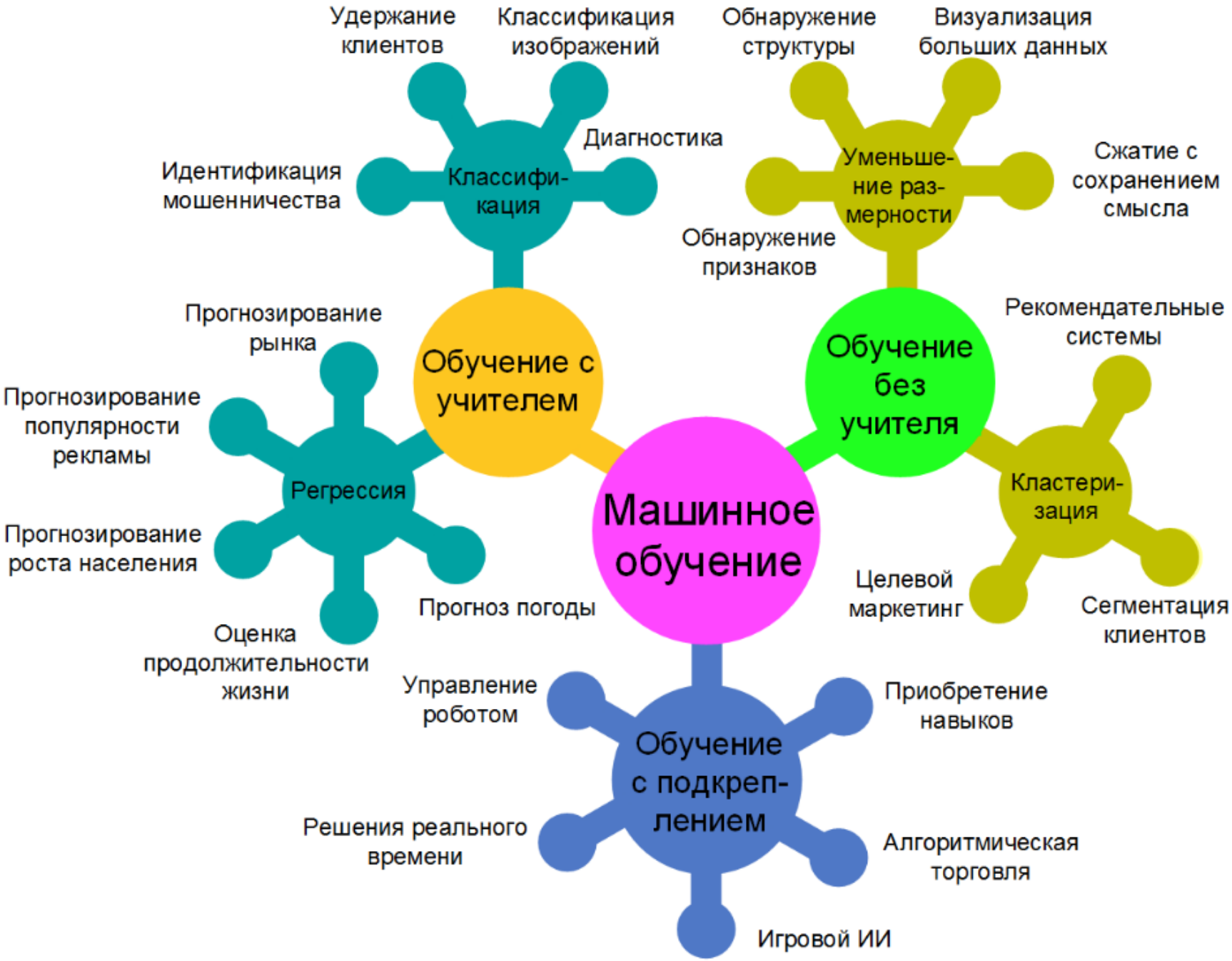
Урок 1. Алгоритм линейной регрессии. Градиентный спуск.

План занятия

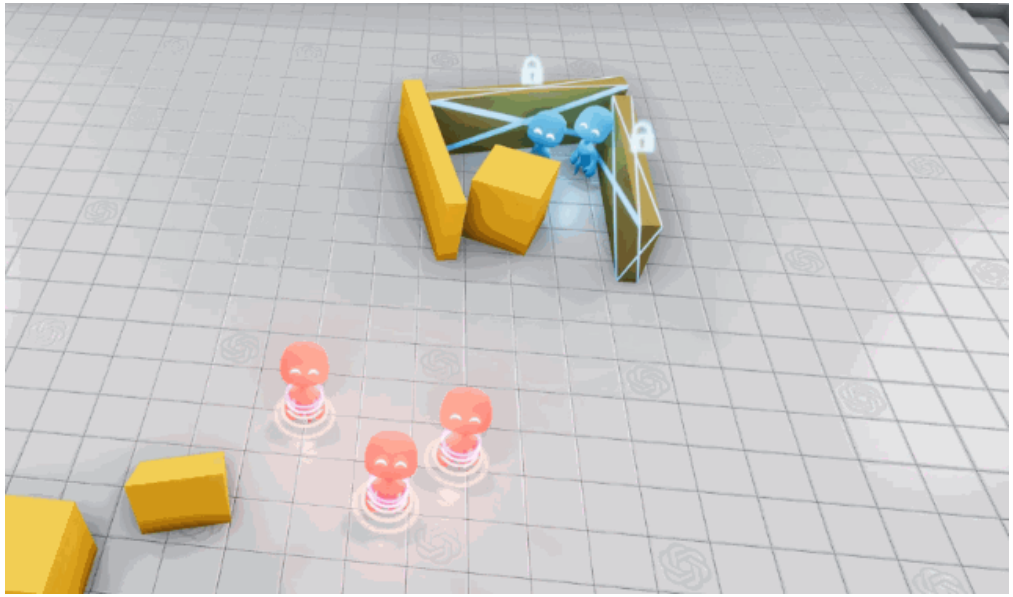
Теоретическая часть

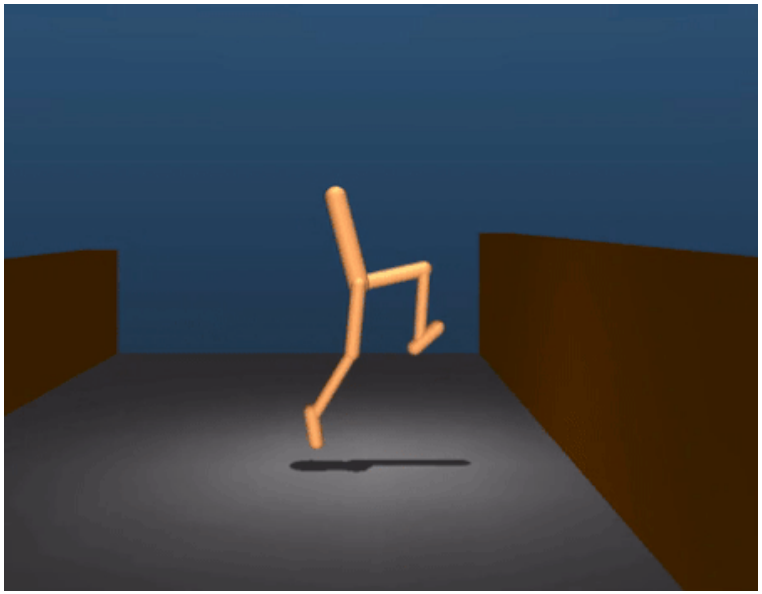
Основные понятия и обозначения

В предыдущих курсах студенты уже знакомились с признаковыми описаниями, основными алгоритмами машинного обучения и Python-библиотеками, используемыми для решения задач в этой области. Однако, перед началом изучения программы курса давайте повторим основные понятия и обозначения, используемые в машинном обучении.


Машинное обучение
Обучение с учителем
Обучение без учителя
Обучение с подкреп-лением
Классифи-кация
Регрессия
Уменьше-ние раз-мерности
Кластери-зация
Удержание клиентов
Классификация изображений
Диагностика
Идентификация мошенничества
Прогнозирование рынка
Прогнозирование популярности рекламы
Прогнозирование роста населения
Оценка продолжительности жизни
Прогноз погоды
Обнаружение структуры
Визуализация больших данных
Сжатие с сохранением смысла
Обнаружение признаков
Рекомендательные системы
Целевой маркетинг
Сегментация клиентов
Управление роботом
Приобретение навыков
Решения реального времени
Алгоритмическая торговля
Игровой ИИ

Обучение с подкреплением - испытуемая система (агент) обучается, взаимодействуя с некоторой средой.





Машинное обучение - дисциплина, заключающаяся в извлечении знаний из известных данных. Машинное обучение - это раздел математики, поэтому в нем мы будем, помимо всего прочего, работать с формулами.

Объект - то, для чего нужно сделать предсказание. Например, в задаче распознавания спам-почты объектом будет являться письмо. Объекты обозначаются буквой x. Множество всех объектов, для которых может потребоваться сделать предсказания, называется *пространством объектов* и обозначается $\mathbb{X}$.

Ответ - то, что нужно предсказать. В том же примере распознавания спама ответом будет является информация о том, является письмо спамом или нет. Ответы обозначаются буквой y (можно сказать, что $y = y(x)$, так как ответ зависит от объекта). *Пространство ответов* - множество всех ответов, с которыми мы можем работать. Оно обозначается $\mathbb{Y}$. В примере задачи распознавания спама оно состоит из двух элементов: $+1$ и -1 (означающие, что письмо является и не является спамом, соответственно).

Для реализации машинного обучения компьютеру нужно "объяснить" объекты, которые в первоначальном виде он понять не может, с помощью сущностей, ему понятных, например, чисел. Для этого вводится понятие *признаков*. Признак - это некая числовая характеристика объекта. Совокупность всех признаков объекта $x = (x^1, x^2, \ldots, x^d)$ называется его *признаковым описанием*. Оно является d-мерным вектором, то есть к нему можно применять все операции, описанные линейной алгеброй.

	symbols	email	feature_3	feature_4	is_spam
0	348	Email_1	425	-3	0
1	230	Email_2	-79	4	1
2	364	Email_3	141	-8	0
3	373	Email_4	466	-6	1
4	253	Email_2	207	-1	1
5	63	Email_5	309	-1	0
6	75	Email_1	364	2	0
7	239	Email_6	101	-7	0

Множество значений i-го признака будем обозначать D_i. Существует множество различных видов признаков:

- *Бинарные признаки* принимают два значения: $D_i = \{0, 1\}$. Примером в задаче кредитного скоринга может служить ответ, выше ли доход клиента определенной установленной суммы. При положительном ответе признак полагается равным 1, при отрицательном - 0.
- *Вещественные признаки* могут принимать в качестве значений все вещественные числа: $D_i = \mathbb{R}$. Примерами могут выступать возраст человека, заработная плата, количество звонков в колл-центр в месяц и т.д.
- *Категориальные признаки* - это такие признаки, значения которых можно сравнивать только на равенство, и нельзя на "больше-меньше". В этом случае D_i - неупорядоченное множетсво. Примерами таких признаков могут выступать город, в котором родился клиент банка, или его образование.
- *Порядковые признаки* - частный случай категориальных признаков. В этом случае D_i - упорядоченное множество. Признаки можно сравнивать между собой, но нельзя определить расстояние между ними. Например, то же образование, но с введенным осмысленным порядком (высшее образование больше среднего профессионального, которое в свою очередь больше среднего и т.д.)

В первой части нашего курса мы рассмотрим алгоритмы *обучения с учителем* или *контролироемого обучения (supervised learning)*. Данный метод заключается в восстановлении общей закономерности по конечному числу известных примеров.

Центральным понятием машинного обучения является *обучающая выборка*. Это примеры, на основе которых мы планируем строить общую закономерность. Она обозначается X и состоит из l пар объектов x_i и известных ответов y_i:

$$X = (x^i, y_i)_{i=1}^l.$$

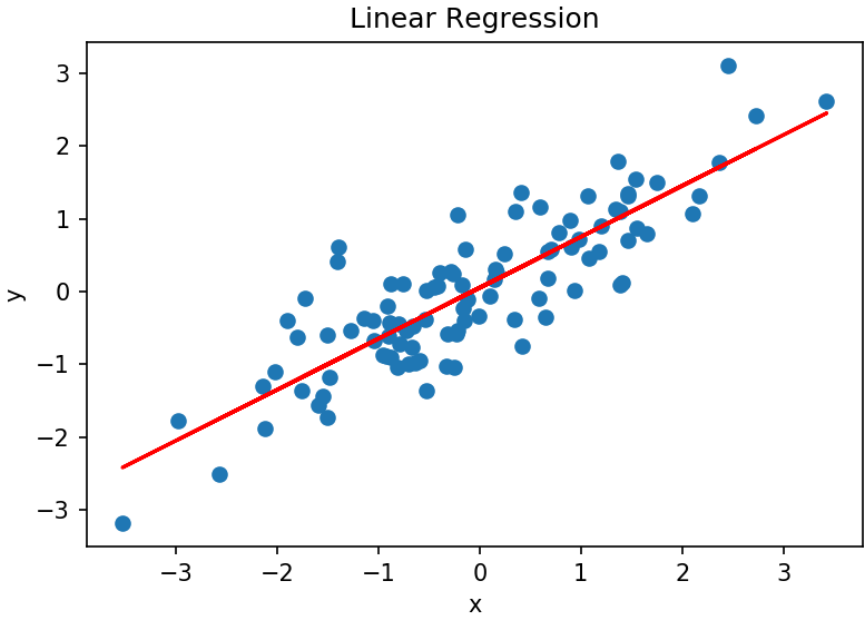
	symbols	email	feature_3	feature_4	is_spam
0	348	Email_1	425	-3	0
1	230	Email_2	-79	4	1
2	364	Email_3	141	-8	0
3	373	Email_4	466	-6	1
4	253	Email_2	207	-1	1
5	63	Email_5	309	-1	0
6	75	Email_1	364	2	0
7	239	Email_6	101	-7	0

Функция, отображающая пространство объектов $\mathbb{X}$ в пространство ответов $\mathbb{Y}$, помогающая нам делать предсказания, называется *алгоритмом* или *моделью* и обозначается $a(x)$. Она принимает на вход объект и выдает ответ.

Для решения определенной задачи не все алгоритмы одинаково хорошо подходят. Для определения наиболее подходящего алгоритма введена характеристика, называемая *функционалом ошибки* $Q(a, X)$. Он принимает на вход алгоритм и выборку и сообщает, насколько хорошо данный алгоритм работает на данной выборке. В примере распознавания спам-писем в качестве такого функционала может выступать доля неправильных ответов (предсказаний). Задача обучения заключается в подборе такого алгоритма, при котором достигается минимум функционала ошибки $Q(a, X) \to min$.

Наиболее подходящий алгоритм при этом выбирается из множества, называемого *семейством алгоритмов* $\mathbb{A}$. Их мы будем рассматривать в данном курсе.

Линейная регрессия. MSE



Ввод [1]:
```
y = ax + a1x1 + a2x2 + b
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_9000/679705834.py in <module>
----> 1 y = ax + a1x1 + a2x2 + b

NameError: name 'ax' is not defined
```

Линейные модели - это такие модели, которые сводятся к суммированию значений признаков с некоторыми весами. Само название модели говорит о том, что зависимость предсказываемой переменной от признаков будет линейной:

$$a(x) = w_0 + \sum_{i=1}^{d} w_i x_i.$$

Линейная регрессия — модель зависимости переменной x от одной или нескольких других переменных (факторов, регрессоров, независимых переменных) с линейной функцией зависимости.

В данном случае параметрами моделей являются веса w_i. Вес w_0 называется *свободным коэффициентом* или *сдвигом*. Оптимизация модели в таком случае заключается в подборе оптимальных значений весов. Сумму в формуле также можно описать как скалярное произведение вектора признаков $x = (x^1, \ldots, x^d)$ на вектор весов $w = (w_1, \ldots, w_d)$:

	for_intercept	symbols	email	feature_3	feature_4	is_spam
0	1	348	Email_1	425	-3	0
1	1	230	Email_2	-79	4	1
2	1	364	Email_3	141	-8	0
3	1	373	Email_4	466	-6	1
4	1	253	Email_2	207	-1	1
5	1	63	Email_5	309	-1	0
6	1	75	Email_1	364	2	0
7	1	239	Email_6	101	-7	0

Обратим внимание, что сдвиг делает модель неоднородной и затрудняет ее дальнейшую оптимизацию. Для устранения этого фактора обычно используют прием, позволяющий упростить запись: к признаковому описанию объекта добавляется еще один признак (константный), на каждом объекте равный единице. В этом случае вес при нем как раз будет по смыслу совпадать со свободным коэффициентом, и сам w_0 будет не нужен. Тогда получим

$$a(x) = \sum_{i=1}^{d+1} w_i x_i = \langle w, x \rangle .$$

За счет простой формы линейные модели достаточно легко обучаются и позволяют работать с зашумленными данными, небольшими выборками, контролируя при этом риск переобучения.

$a(x)$	y	$a(x) - y$	$\lvert a(x) - y \rvert$	$(a(x) - y)^2$
11	10	1	1	1
9	10	-1	1	1
20	10	10	10	100
1	10	-9	9	81

Для обучения модели необходимо иметь возможность измерять точность линейного алгоритма на выборке (обучающей или тестовой).

В качестве меры ошибки можно взять абсолютное отклонение истинного значения от прогноза $Q(a, y) = a(x) - y$, но тогда минимизация функционала ошибки (в которой и состоит задача обучения) будет достигаться при принятии им отрицательного значения. Например, если истинное значение ответа равно 10, а алгоритм $a(x)$ выдает ответ 11, отклонение будет равно 1, а при значении предсказания равном 1, отклонение будет равно $1 - 10 = -9$. Значение ошибки во втором случае ниже, однако разница истинного значения и предсказания нашей модели больше. Таким образом, такой функционал ошибки не поддается минимизации.

Логичным кажется решение использовать в качестве функционала ошибки модуль отклонения $Q(a, y) = |a(x) - y|$. Соответствующий функционал ошибки называется средним абсолютным отклонением (mean absolute error, MAE):

$$Q(a, x) = \frac{1}{l} \sum_{i=1}^{l} |a(x_i) - y_i|.$$

Однако, как мы далее увидим, зачастую методы оптимизации включают в себя дифференцирование, а функция модуля не является гладкой - она не имеет производной в нуле, поэтому ее оптимизация бывает затруднительной.

Поэтому сейчас основной способ измерить отклонение - посчитать квадрат разности $Q(a, y) = (a(x) - y)^2$. Такая функция является гладкой и имеет производную в каждой точке, а ее минимум достигается при равенстве истинного ответа y и прогноза $a(x)$.

Основанный на этой функции функционал ошибки называется *среднеквадратичным отклонением* (mean squared error, MSE):

$$Q(a, x) = \frac{1}{l} \sum_{i=1}^{l} (a(x_i) - y_i)^2.$$

Метод наименьших квадратов

Как уже говорилось ранее, обучение модели регрессии заключается в минимизации функционала ошибки. Таким образом, в случае использования среднеквадратичной ошибки получаем задачу оптимизации

$$Q(w, x) = \frac{1}{l}\sum_{i=1}^{l}(\langle w, x_i\rangle - y_i)^2 \to \min_w.$$

Способ вычисления весов путем минимизации среднеквадратичного отклонения называется **методом наименьших квадратов**.

Заметим, что здесь мы переписали выражение функционала ошибки, заменив $a(x)$ на скалярное призведение $\langle w, x\rangle$, после чего мы уже имеем функцию, а не функционал ошибки, так как Q зависит не от некоторой функции $a(x)$, а от вектора весов w, и оптимизировать нужно именно по нему, что гораздо проще.

Имеет смысл переписать имеющиеся соотношения в матричном виде. В матрицу "объекты-признаки" впишем по строкам d признаков для всех l объектов из обучающей выборки:

$$X = \begin{pmatrix} x_{11} & \dots & x_{1d} \\ \dots & \dots & \dots \\ x_{l1} & \dots & x_{ld} \end{pmatrix},$$

и составим вектор ответов y из истинных ответов для данной выборки:

$$y = \begin{pmatrix} y_1 \\ \dots \\ y_l \end{pmatrix}.$$

Помня, что w - вектор параметров, переписанная в матричном виде задача будет выглядеть следующим образом:

$$Q(w, X) = \frac{1}{l}||Xw - y||^2 \to \min_w,$$

где используется евклидова (L_2) норма:

$$||x|| = \sqrt{\sum_{i=1}^{l} x_i^2}$$

$$||Xw - y|| = \sqrt{\sum_{i=1}^{l} (X_i w - y_i)^2}$$

$$Q(w, X) = \frac{1}{l} \sqrt{\sum_{i=1}^{l} (X_i w - y_i)^2}^2 = \frac{1}{l} \sum_{i=1}^{l} (X_i w - y_i)^2$$

$\nabla_w Q(w, X) = \frac{2}{l} X^T (Xw - y).$

$\frac{2}{l} X^T (Xw - y) = 0$

$X^T (Xw - y) = 0$

$X^T Xw - X^T y = 0$

$w = \frac{X^T y}{X^T X}$

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

Продифференцировав данную функцию по вектору w и приравняв к нулю, можно получить явную анатилическую формулу для решения задачи минимизации (ссылка (см. пункт 1.2) (https://habr.com/ru/company/ods/blog/323890/#metod-naimenshih-kvadratov) на подробный вывод аналитической формулы решения уравнения линейной регрессии):

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

Это решение называется *нормальным уравнением* линейной регрессии. Наличие аналитического решения кажется положительным фактором, однако, у него есть некоторые минусы, среди которых вычислительная сложность операции (обращение матрицы $X^T X$ будет иметь кубическую сложность от количества признаков d^3), а также тот факт, что матрица $X^T X$ может быть вырожденной и поэтому необратимой. Тогда найти решение будет невозможно.

Более удобным подходом будет разработка решения с помощью численных методов оптимизации, одним из которых является *градиентный спуск*.

Градиентный спуск

Среднеквадратичная ошибка имеет один минимум и непрерывна на всей области значений (то есть является выпуклой и гладкой), а значит в каждой ее точке можно посчитать частные производные.

Вспомним, что *градиентом* функции f называется n-мерный вектор из частных производных.

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

При этом известно, что **градиент задает направление наискорейшего роста функции**. Значит, антиградиент будет показывать направление ее скорейшего убывания, что будет полезно нам в нашей задаче минимизации функционала ошибки.

Градиентный спуск — метод нахождения локального экстремума функции (минимума или максимума) с помощью движения вдоль градиента.

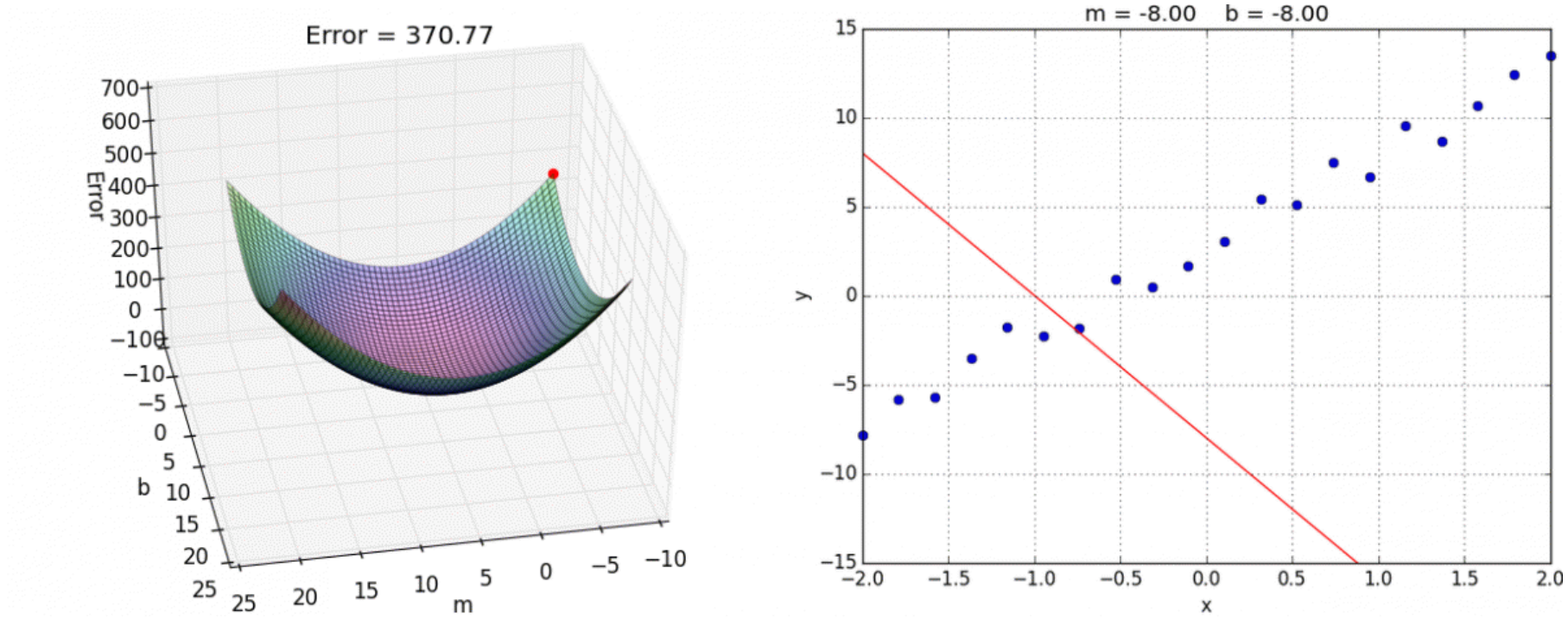
Ввод [ ]:
```
df(x1, x2, x3) = (df(x1), df(x2), df(x3))
```

Для решения задачи нам требуется определить некоторую стартовую точку и итерационно сдвигаться от нее в сторону антиградиента с определенным *шагом* η_k, на каждом шагу пересчитывая градиент в точке, в которой мы находимся. Таким образом, имея начальный вектор весов w^0, k-й шаг градиентного спуска будет иметь вид

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

Итерации следует продолжать, пока не наступает сходимость. Она определяется разными способами, но в даннном случае удобно определять как ситуацию, когда векторы весов от шага к шагу изменяются незначительно, то есть норма отклонения вектора весов на текущем шаге от предыдущего не привышает заданное значение ε:

$$||w^k - w^{k-1}|| < \varepsilon.$$



Начальный вектор весов w_0 также можно определять различными способами, обычно его берут нулевым или состоящим из случайных небольших чисел.

В случае многомерной регрессии (при количестве признаков больше 1) при оптимизации функционала ошибки

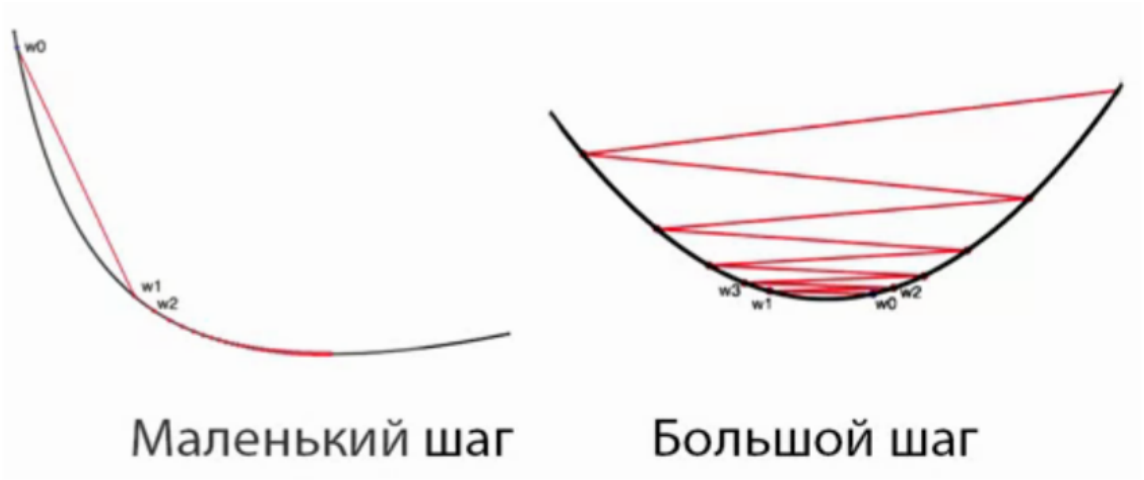
$$Q(w, X) = \frac{1}{l}||Xw - y||^2 \to \min_w$$

формула вычисления градиента принимает вид

$$\nabla_w Q(w, X) = \frac{2}{l}X^T(Xw - y).$$

Ввод [ ]:
```
1. w0 = 0
2. k = 1:
        w1 = w0 - grad(w0)
```

1. Инициализация w
2. Цикл по k = 1,2,3,...:
   - $w^k = w^{k-1} - \eta_k \nabla Q(w^{k-1}, X)$
   - Если $||w^k - w^{k-1}|| < \epsilon$, то завершить.



Практическая часть

Смоделируем работу градиентного спуска при помощи Numpy.

Ввод [2]:
```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.axes3d import Axes3D
```

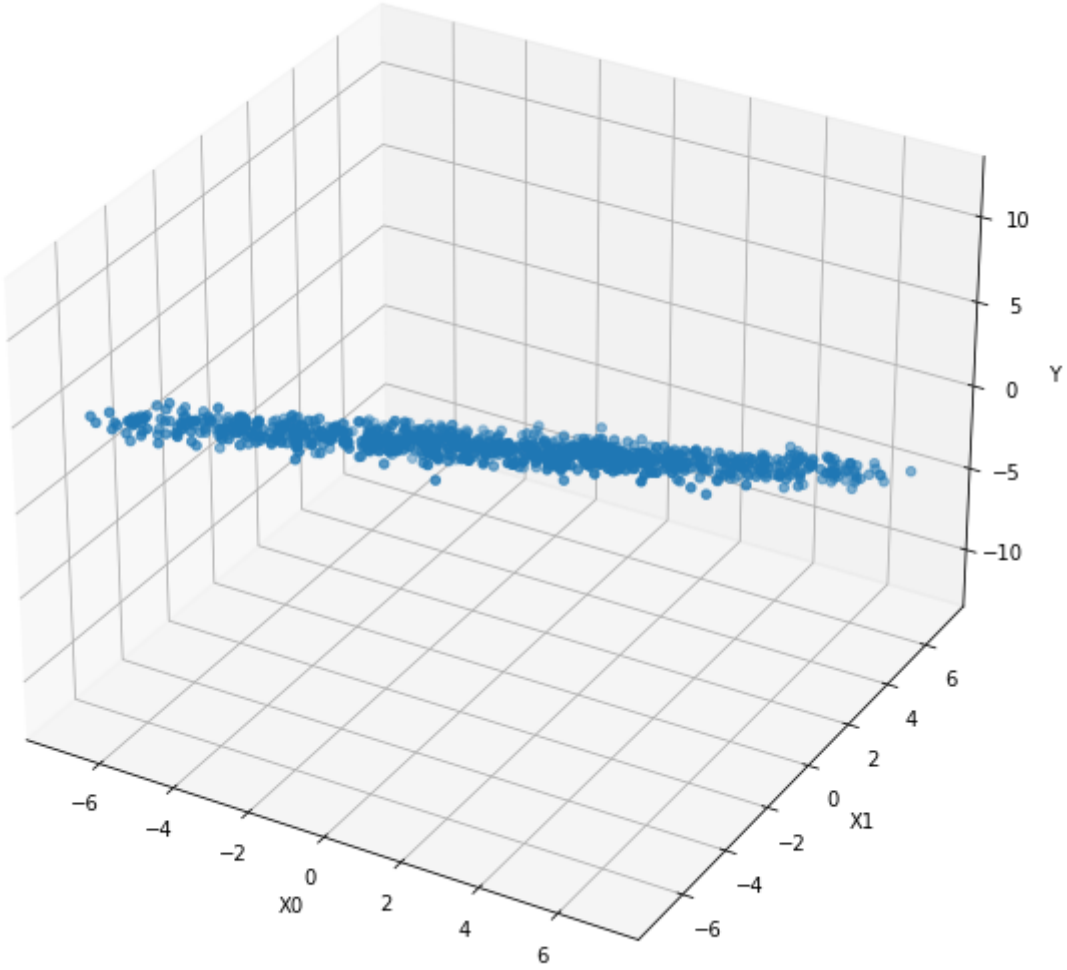
Ввод [3]:
```
np.random.seed(1234)
# Возьмем 2 признака и 1000 объектов
n_features = 2
n_objects = 1000

# сгенерируем вектор истинных весов
w_true = np.random.normal(size=(n_features))

# сгенерируем матрицу X, вычислим Y с добавлением случайного шума
X = np.random.uniform(-7, 7, (n_objects, n_features))
Y = X.dot(w_true) + np.random.normal(0, 0.5, size=(n_objects))

# возьмем нулевые начальные веса
w = np.zeros(n_features)
```

Ввод [4]:
```
display(w, w_true, X, Y)
```

```
array([0., 0.])

array([ 0.47143516, -1.19097569])

array([[-0.87181165,  3.99502017],
       [ 3.91966131, -3.18370353],
       [-3.12950043,  4.22621049],
       ...,
       [ 3.51597757,  5.11940305],
       [ 6.35903104,  5.31390661],
       [ 5.12347492,  2.96879971]])

array([-5.71937308e+00,  6.42200487e+00, -6.22754907e+00, -3.61309248e+00,
       -1.24952278e+00, -2.10667534e+00, -1.87408546e+00,  8.06711862e+00,
       -4.75602542e+00, -2.03931948e+00, -5.77773483e-01,  8.32648436e-01,
       -5.55413438e+00, -1.84424278e+00,  3.29500371e+00,  8.02004094e+00,
       -1.64093596e+00, -8.35192303e+00, -6.93743921e+00,  2.59477266e+00,
       -5.89712012e+00,  1.15446630e-01, -4.28621134e+00, -1.65799764e+00,
       -4.20035318e+00,  8.65844888e+00, -5.58161780e+00, -5.24268607e+00,
       -1.93540562e+00, -4.29251150e+00,  6.18523295e+00,  6.04222590e+00,
```

Ввод [5]:

```
fig = plt.figure(figsize=(15,10))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(X[:, 0], X[:, 1], Y)

ax.set_xlabel('X0')
ax.set_ylabel('X1')
ax.set_zlabel('Y')
plt.show()
```


10
5
0
−5
−10
Y
−6
−4
−2
0
2
4
6
X0
6
4
2
0
−2
−4
−6
X1

Ввод [6]:
```
# реализуем функцию, определяющую среднеквадратичную ошибку
def mserror(X, w, y):
    y_pred = X.dot(w)
    return (np.sum((y_pred - y)**2)) / len(y)
```

Реализуем функцию, вычисляющую вектор весов по нормальному уравнению линейной регрессии, и применим ее.

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

Ввод [7]:
```
normal_eq_w = np.linalg.inv(np.dot(X.T, X)) @ X.T @ Y
print(f'Веса {normal_eq_w}')
print(f'В случае использования нормального уравнения функционал ошибки составляет ', end='')
print(f'{round(mserror(X, normal_eq_w, Y), 4)}')
```

```
Веса [ 0.46622325 -1.18317886]
В случае использования нормального уравнения функционал ошибки составляет 0.2413
```

Ввод [8]:
```
w_true
```

Out[8]:
```
array([ 0.47143516, -1.19097569])
```

Обучим линейную регрессию путем градиентного спуска и получим графики изменения весов и ошибки

Ввод [9]:
```
w
```

Out[9]:
```
array([0., 0.])
```

Ввод [10]:

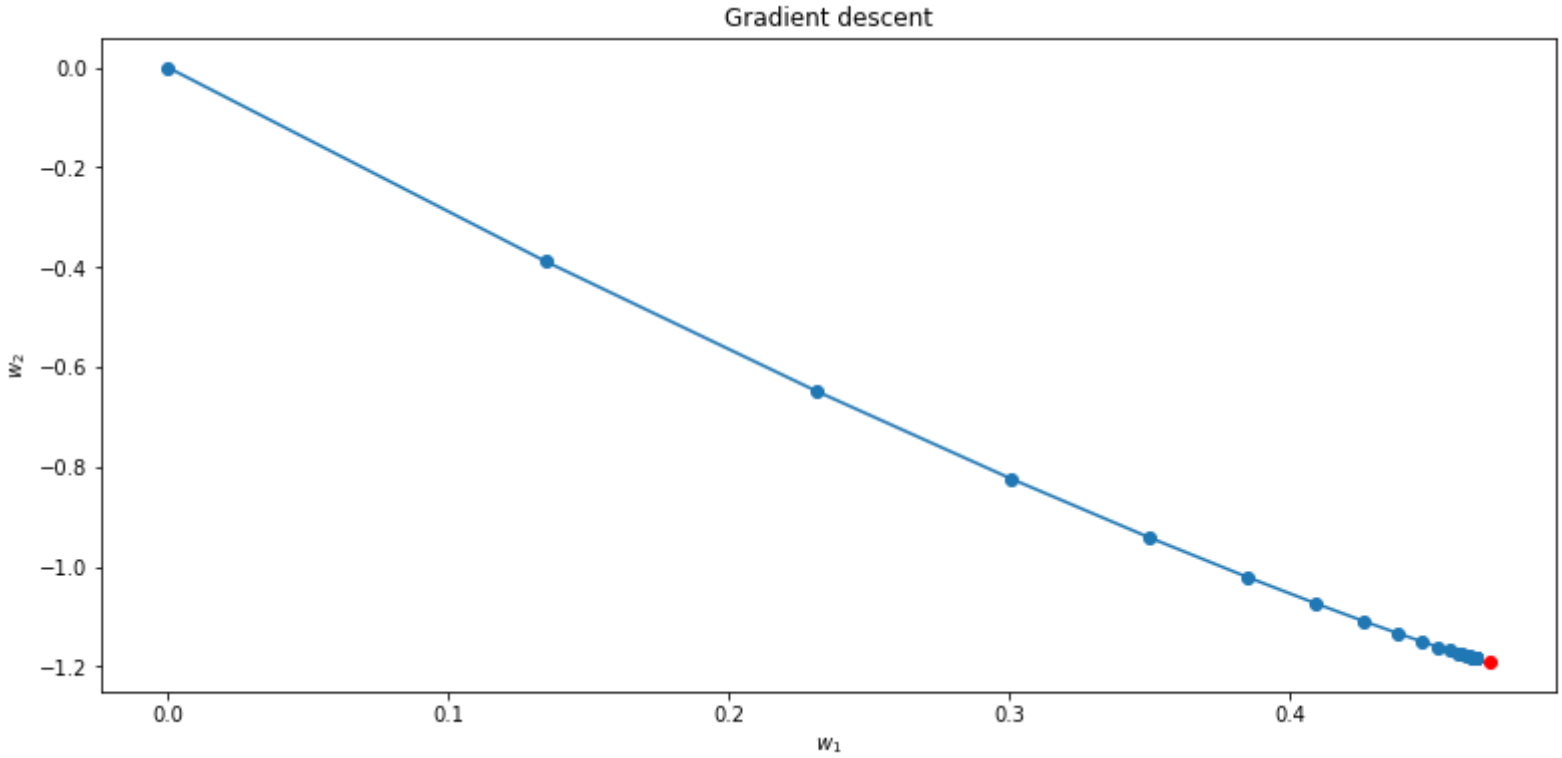
```
# возьмем нулевые начальные веса
w = np.zeros(n_features)

# список векторов весов после каждой итерации
w_list = [w.copy()]

# список значений ошибок после каждой итерации
errors = []

# шаг градиентного спуска
eta = 0.01

# максимальное число итераций
max_iter = 1e4

# критерий сходимости (разница весов, при которой алгоритм останавливается)
min_weight_dist = 1e-8

# зададим начальную разницу весов большим числом
weight_dist = np.inf

# счетчик итераций
iter_num = 0

# ход градиентного спуска
while weight_dist > min_weight_dist and iter_num < max_iter:
    y_pred = np.dot(X, w)
    dQ = 2 / Y.shape[0] * np.dot(X.T, y_pred - Y)
    new_w = w - eta * dQ
    weight_dist = np.linalg.norm(new_w - w, ord=2)
    error = mserror(X, new_w, Y)

    w_list.append(new_w.copy())
    errors.append(error)

    print(f'Iter {iter_num}: error - {error}, weights: {new_w}')

    iter_num += 1
    w = new_w

w_list = np.array(w_list)
```

```
w_pred = w_list[-1]

print(f'В случае использования градиентного спуска функционал ошибки составляет {round(errors[-1], 4)}')
```

```
Iter 0: error - 12.198224910563487, weights: [ 0.13477638 -0.38857461]
Iter 1: error - 5.721927562708119, weights: [ 0.23166602 -0.64911175]
Iter 2: error - 2.757139097172467, weights: [ 0.30088498 -0.82395109]
Iter 3: error - 1.3976931456804218, weights: [ 0.350075   -0.94137792]
Iter 4: error - 0.7734530790003209, weights: [ 0.38487408 -1.02030733]
Iter 5: error - 0.4864474154164298, weights: [ 0.40939674 -1.07340045]
Iter 6: error - 0.3543442315442369, weights: [ 0.42661932 -1.10914011]
Iter 7: error - 0.2934800582148447, weights: [ 0.4386792  -1.13321479]
Iter 8: error - 0.26541385069282625, weights: [ 0.447102   -1.14944239]
Iter 9: error - 0.2524619639743879, weights: [ 0.45297107 -1.16038742]
Iter 10: error - 0.24648103454886586, weights: [ 0.45705232 -1.16777387]
Iter 11: error - 0.24371756626897814, weights: [ 0.45988515 -1.17276152]
Iter 12: error - 0.24244007305388324, weights: [ 0.46184823 -1.17613117]
Iter 13: error - 0.2418492559727374, weights: [ 0.4632066  -1.17840885]
Iter 14: error - 0.24157590963593276, weights: [ 0.46414529 -1.17994914]
Iter 15: error - 0.241449401703911, weights: [ 0.46479317 -1.18099124]
Iter 16: error - 0.241390835409215, weights: [ 0.46523987 -1.18169657]
Iter 17: error - 0.241363715587635, weights: [ 0.46554754 -1.18217416]
Iter 18: error - 0.241351154689716, weights: [ 0.46575927 -1.18249766]
Iter 19: error - 0.2413453358438324, weights: [ 0.46590486 -1.18271686]
Iter 20: error - 0.24134263981533452, weights: [ 0.46600489 -1.18286544]
Iter 21: error - 0.24134139049412393, weights: [ 0.46607357 -1.18296618]
Iter 22: error - 0.24134081149547823, weights: [ 0.4661207  -1.18303451]
Iter 23: error - 0.2413405431293834, weights: [ 0.46615302 -1.18308086]
Iter 24: error - 0.24134041873000472, weights: [ 0.46617518 -1.18311232]
Iter 25: error - 0.2413403610608135, weights: [ 0.46619035 -1.18313367]
Iter 26: error - 0.24134033432459853, weights: [ 0.46620075 -1.18314816]
Iter 27: error - 0.2413403219285762, weights: [ 0.46620786 -1.183158  ]
Iter 28: error - 0.24134031618096188, weights: [ 0.46621273 -1.18316469]
Iter 29: error - 0.24134031351586688, weights: [ 0.46621606 -1.18316923]
Iter 30: error - 0.2413403122800478, weights: [ 0.46621834 -1.18317231]
Iter 31: error - 0.24134031170697204, weights: [ 0.46621989 -1.18317441]
Iter 32: error - 0.24134031144121676, weights: [ 0.46622096 -1.18317584]
Iter 33: error - 0.2413403113179735, weights: [ 0.46622169 -1.1831768 ]
Iter 34: error - 0.2413403112608185, weights: [ 0.46622218 -1.18317746]
Iter 35: error - 0.24134031123431202, weights: [ 0.46622252 -1.18317791]
Iter 36: error - 0.24134031122201902, weights: [ 0.46622275 -1.18317822]
Iter 37: error - 0.24134031121631766, weights: [ 0.46622291 -1.18317842]
Iter 38: error - 0.24134031121367355, weights: [ 0.46622302 -1.18317856]
Iter 39: error - 0.24134031121244717, weights: [ 0.46622309 -1.18317866]
Iter 40: error - 0.24134031121187843, weights: [ 0.46622314 -1.18317872]
```

```
Iter 41: error - 0.24134031121161462, weights: [ 0.46622318 -1.18317877]
Iter 42: error - 0.24134031121149224, weights: [ 0.4662232 -1.1831788]
Iter 43: error - 0.2413403112114355, weights: [ 0.46622322 -1.18317882]
Iter 44: error - 0.2413403112114092, weights: [ 0.46622323 -1.18317883]
Iter 45: error - 0.24134031121139696, weights: [ 0.46622323 -1.18317884]
Iter 46: error - 0.24134031121139135, weights: [ 0.46622324 -1.18317885]
В случае использования градиентного спуска функционал ошибки составляет 0.2413
```

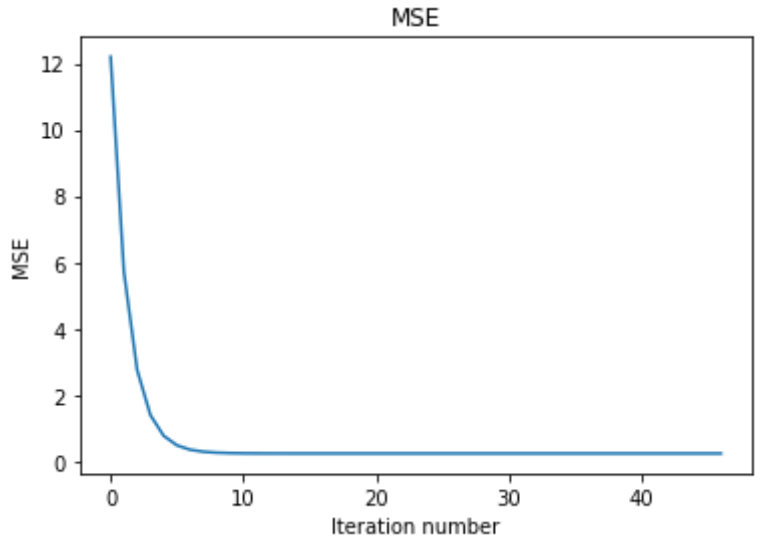
Ввод [11]:
```
# Визуализируем изменение весов (красной точкой обозначены истинные веса, сгенерированные вначале)
plt.figure(figsize=(13, 6))
plt.title('Gradient descent')
plt.xlabel(r'$w_1$')
plt.ylabel(r'$w_2$')

plt.scatter(w_list[:, 0], w_list[:, 1])
plt.scatter(w_true[0], w_true[1], c='r')
plt.plot(w_list[:, 0], w_list[:, 1])

plt.show()
```



После каждой итерации значения искомых весов приближаются к истинным, однако, не становятся им равны из-за шума, добавленного в вектор ответов.

Ввод [12]:
```
# Визуализируем изменение функционала ошибки
plt.plot(range(len(errors)), errors)
plt.title('MSE')
plt.xlabel('Iteration number')
plt.ylabel('MSE')
```

Out[12]:
```
Text(0, 0.5, 'MSE')
```



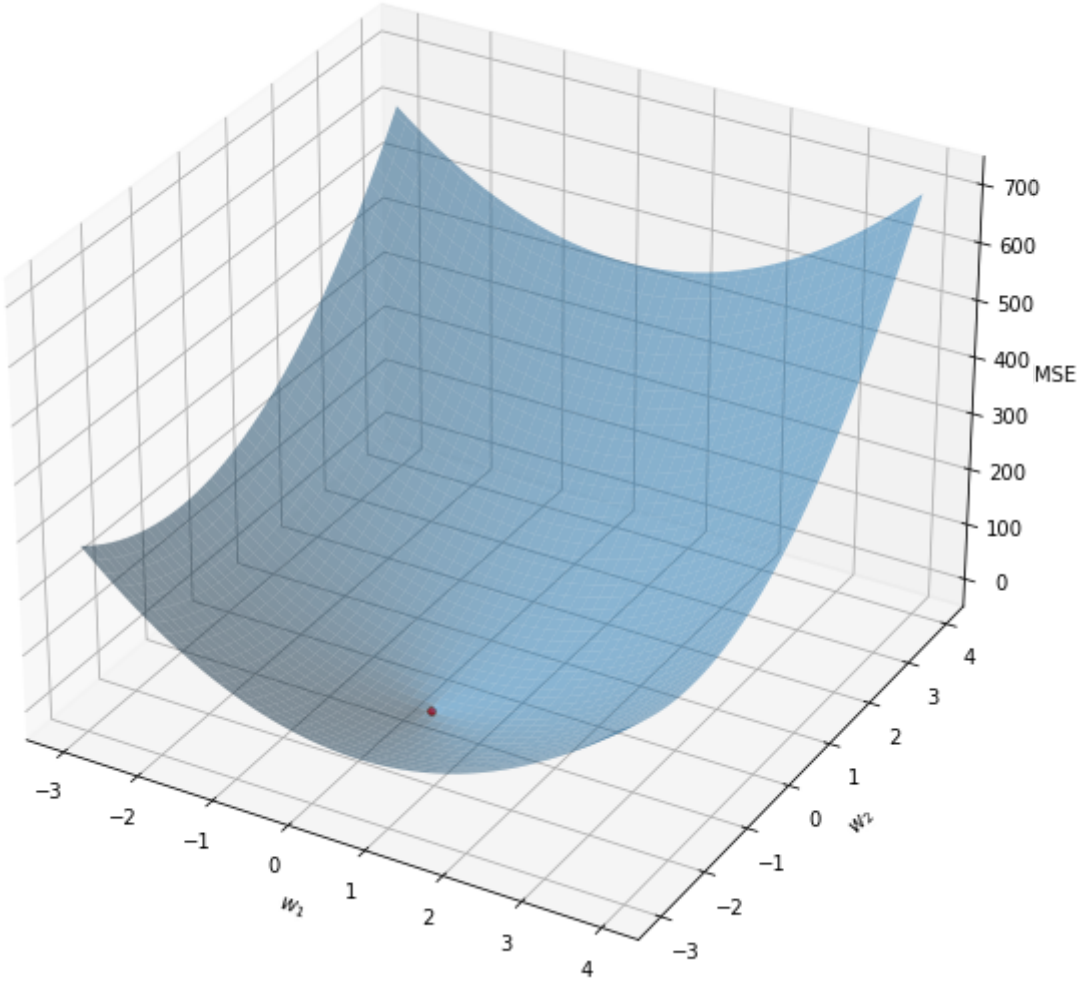
Видно, что изменение монотонно и начинается с высокой точки, после определенного количества итераций выходя на асимптоту.

Ввод [13]:
```
np.arange(-3.0, 4.0, 0.05).shape
```

Out[13]:
```
(140,)
```

Ввод [14]:

```
def mse_manual(w1, w2, y_pred):
    w = np.array([w1, w2])
    y = X.dot(w)
    return (sum((y - y_pred)**2)) / len(y)

fig = plt.figure(figsize=(15, 10))
ax = fig.add_subplot(111, projection='3d')
w1 = w2 = np.arange(-3.0, 4.0, 0.05)
w1, w2 = np.meshgrid(w1, w2)

zs = np.array([mse_manual(i, j, Y) for i, j in zip(np.ravel(w1), np.ravel(w2))])
Z = zs.reshape(w1.shape)

ax.scatter(w_true[0], w_true[1], mse_manual(w_true[0], w_true[1], Y), c='r', s=10)
ax.scatter(w_pred[0], w_pred[1], mse_manual(w_pred[0], w_pred[1], Y), c='g', s=10)
ax.plot_surface(w1, w2, Z, alpha=.5)

ax.set_xlabel(r'$w_1$')
ax.set_ylabel(r'$w_2$')
ax.set_zlabel('MSE')

plt.show()
```



Очень важно при использовании метода градиентного спуска правильно подбирать шаг. Если длина шага будет слишком мала, то метод будет слишком медленно приближаться к правильному ответу, и потребуется очень большое количество итераций для достижения сходимости. Если же длина наоборот будет слишком большой, появится вероятность "перепрыгивания" алгоритма через минимум функции или вообще отсутствия сходимости градиентного спуска.

Применяется методика использования переменного размера шага: на начальных этапах берется большой шаг, который с увеличением количества итераций снижается. Одна из таких методик - вычисление на каждой итерации размера шага по формуле

$$\eta_k = \frac{c}{k},$$

Пример задачи

Задача: предсказание баллов ЕГЭ ученика в зависимости от кол-ва лет стажа его репетитора

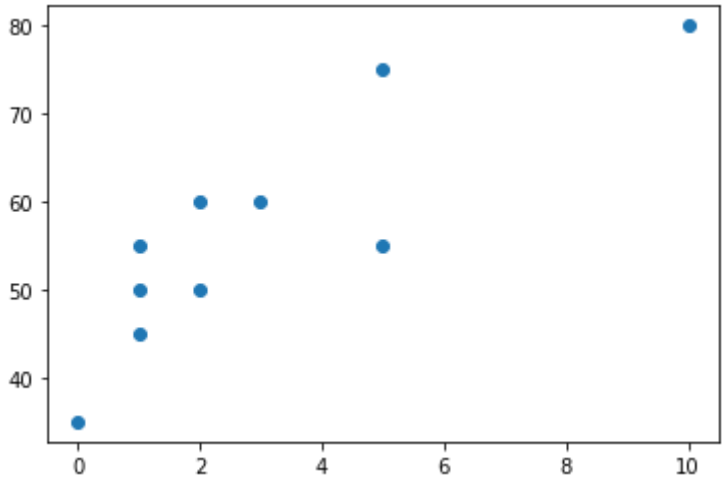
Ввод [15]:
```
X = np.array([[ 1,  1],
              [ 1,  1],
              [ 1,  2],
              [ 1,  5],
              [ 1,  3],
              [ 1,  0],
              [ 1,  5],
              [ 1, 10],
              [ 1,  1],
              [ 1,  2]])
```

Ввод [16]:
```
X.shape
```

Out[16]:
```
(10, 2)
```

Ввод [17]:
```
y = [45, 55, 50, 55, 60, 35, 75, 80, 50, 60]
```

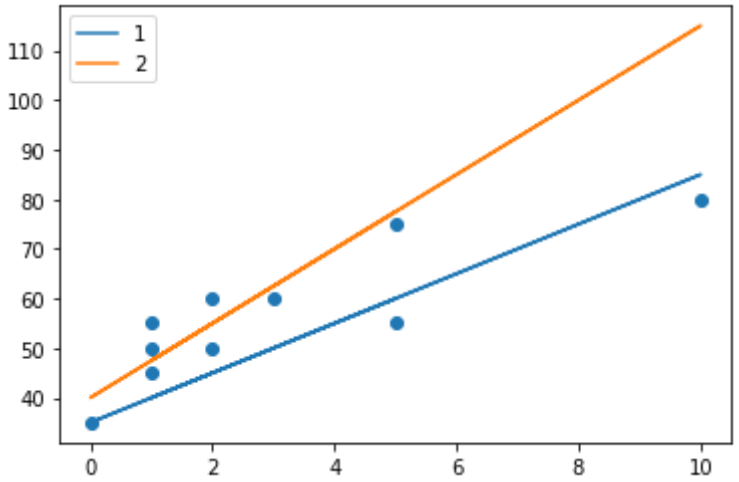
Ввод [18]:
```
plt.scatter(X[:, 1], y);
```



Уравнение прямой: $y = a * x + b$

Ввод [19]:
```
y_pred1 = 5 * X[:, 1] + 35 * X[:, 0]
y_pred2 = 7.5 * X[:, 1] + 40 * X[:, 0]
```

Ввод [20]:
```
plt.scatter(X[:, 1], y)
plt.plot(X[:, 1], y_pred1, label='1')
plt.plot(X[:, 1], y_pred2, label='2')
plt.legend()
plt.show()
```



Отклонение

Ввод [21]:
```
err1 = np.sum(y - y_pred1)
err2 = np.sum(y - y_pred2)
err1, err2
```

Out[21]:
```
(65, -60.0)
```

MAE (Mean Absolute Error)

Ввод [22]:
```
mae_1 = np.mean(np.abs(y - y_pred1))
mae_2 = np.mean(np.abs(y - y_pred2))
mae_1, mae_2
```

Out[22]:
```
(8.5, 9.0)
```

MSE (Mean Squared Error)

Ввод [23]:
```
mse_1 = np.mean((y - y_pred1)**2)
mse_2 = np.mean((y - y_pred2)**2)
mse_1, mse_2
```

Out[23]:
```
(97.5, 188.75)
```

Метод наименьших квадратов (МНК)

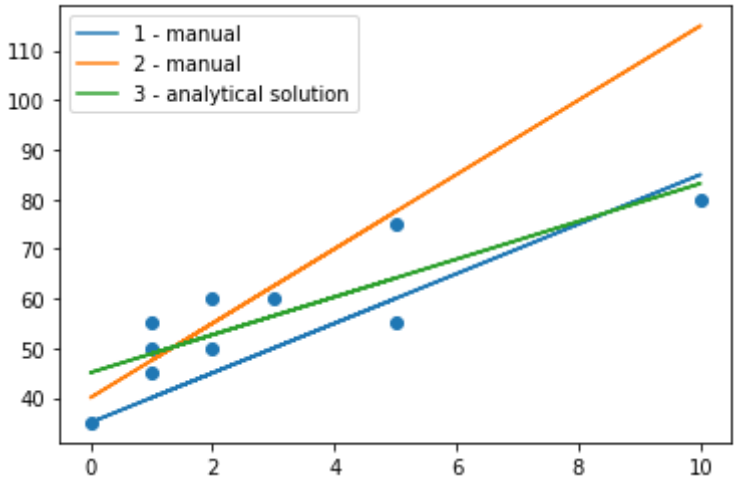
$$w = (X^T X)^{-1} X^T y.$$

Ввод [24]:
```
W_analytical = np.linalg.inv(np.dot(X.T, X)) @ X.T @ y
W_analytical
```

Out[24]:
```
array([45.0625,  3.8125])
```

Ввод [25]:
```
y_pred_analytical = W_analytical[0] * X[:, 0] + W_analytical[1] * X[:, 1]
y_pred_analytical = X @ W_analytical
```

Ввод [26]:
```
plt.scatter(X[:, 1], y)
plt.plot(X[:, 1], y_pred1, label='1 - manual')
plt.plot(X[:, 1], y_pred2, label='2 - manual')
plt.plot(X[:, 1], y_pred_analytical, label='3 - analytical solution')
plt.legend()
plt.show()
```



Ввод [27]:
```
def calc_mae(y, y_pred):
    err = np.mean(np.abs(y - y_pred))
    return err

def calc_mse(y, y_pred):
    err = np.mean((y - y_pred)**2)
    return err
```

Ввод [28]:
```
calc_mae(y, y_pred1), calc_mse(y, y_pred1)
```

Out[28]:
```
(8.5, 97.5)
```

Ввод [29]:
```
calc_mae(y, y_pred2), calc_mse(y, y_pred2)
```

Out[29]:
```
(9.0, 188.75)
```

Ввод [30]:
```
calc_mae(y, y_pred_analytical), calc_mse(y, y_pred_analytical)
```

Out[30]:
```
(5.7875, 43.96875)
```

Градиентный спуск

$$\nabla_w Q(w, X) = \frac{2}{l} X^T (Xw - y).$$

1. Инициализация w
2. Цикл по k = 1,2,3,...:
   - $w^k = w^{k-1} - \eta_k \nabla Q(w^{k-1}, X)$
   - Если $||w^k - w^{k-1}|| < \epsilon$, то завершить.

Ввод [31]:
```
W = np.random.normal(size=(X.shape[1]))
W
```

Out[31]:
```
array([-0.03510077, -0.82561501])
```

Ввод [32]:
```
eta = 0.02 # величина шага
```

Ввод [33]:
```
X.shape,  W.shape
```

Out[33]:
```
((10, 2), (2,))
```

Ввод [34]:
```
n = len(y)
dQ = 2/n * X.T @ (X @ W - y) # градиент функции ошибки
dQ
```

Out[34]:
```
array([-118.02389157, -428.28151478])
```

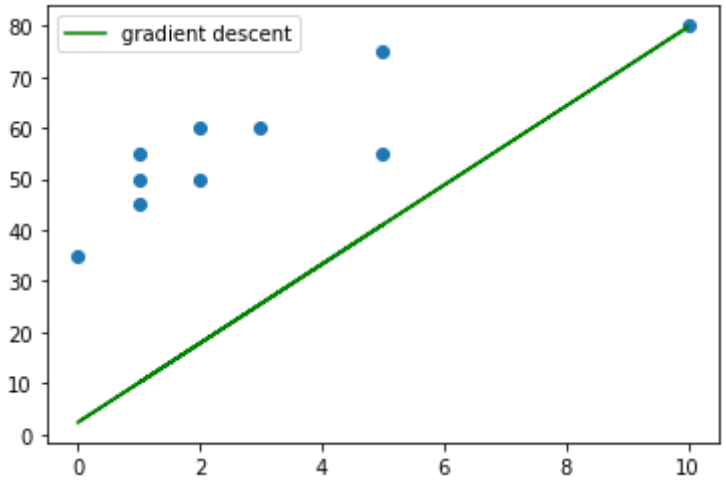
Ввод [35]:
```
grad = eta * dQ
grad
```

Out[35]:
```
array([-2.36047783, -8.5656303 ])
```

Ввод [36]:
```
print(f'previous weights', W)
W = W - grad
print(f'new weights', W)
```

```
previous weights [-0.03510077 -0.82561501]
new weights [2.32537706 7.74001529]
```

Ввод [37]:
```
y_pred_grad = X @ W
plt.scatter(X[:, 1], y)
plt.plot(X[:, 1], y_pred_grad, label='gradient descent', c='g')
plt.legend()
plt.show()
```



Домашнее задание

1. Подберите скорость обучения (eta) и количество итераций

Ввод [43]:

```
n = X.shape[0]

eta = 1e-1
n_iter = 220

W = np.array([1, 0.5])
print(f'Number of objects = {n} \
       \nLearning rate = {eta} \
       \nInitial weights = {W} \n')

for i in range(n_iter):
    y_pred = np.dot(X, W)
    err = calc_mse(y, y_pred)
    for k in range(W.shape[0]):
        W[k] -= eta * (1/n * 2 * X[:, k] @ (y_pred - y))
    if i % 10 == 0:
        eta /= 1.1
        print(f'Iteration #{i}: W_new = {W}, MSE = {round(err, 2)}')
```

```
Number of objects = 10
Learning rate = 0.1
Initial weights = [1.  0.5]

Iteration #0: W_new = [11.8 38.2], MSE = 3047.75
Iteration #10: W_new = [12651.73553914 69617.0969639 ], MSE = 18310954068.05
Iteration #20: W_new = [ 7732434.81888021 42641607.37852186], MSE = 9128819654907568.0
Iteration #30: W_new = [1.06344502e+09 5.86454589e+09], MSE = 2.327920364266843e+20
Iteration #40: W_new = [3.00127077e+10 1.65510116e+11], MSE = 2.545133529815933e+23
Iteration #50: W_new = [1.55345341e+11 8.56677968e+11], MSE = 9.572295620500074e+24
Iteration #60: W_new = [1.27742291e+11 7.04456313e+11], MSE = 9.351480126475957e+24
Iteration #70: W_new = [1.38141953e+10 7.61806995e+10], MSE = 1.640858952828383e+23
Iteration #80: W_new = [1.51674189e+08 8.36432543e+08], MSE = 3.125533537874439e+19
Iteration #90: W_new = [116395.49988139 641638.79864291], MSE = 31317286806394.04
Iteration #100: W_new = [48.41966454 22.99883908], MSE = 53518.86
Iteration #110: W_new = [44.9771605   3.82798314], MSE = 43.97
Iteration #120: W_new = [44.99895429  3.82402303], MSE = 43.97
Iteration #130: W_new = [45.01387996  3.82131649], MSE = 43.97
Iteration #140: W_new = [45.024372    3.81941392], MSE = 43.97
Iteration #150: W_new = [45.03192427  3.81804444], MSE = 43.97
Iteration #160: W_new = [45.03747853  3.81703726], MSE = 43.97
Iteration #170: W_new = [45.04164375  3.81628196], MSE = 43.97
Iteration #180: W_new = [45.04482301  3.81570545], MSE = 43.97
Iteration #190: W_new = [45.04728894  3.81525829], MSE = 43.97
Iteration #200: W_new = [45.04922963  3.81490638], MSE = 43.97
Iteration #210: W_new = [45.05077727  3.81462574], MSE = 43.97
```

2*. В этом коде мы избавляемся от итераций по весам, но тут есть ошибка, исправьте ее

Ввод [44]:

```
n = X.shape[0]

eta = 1e-2
n_iter = 651

W = np.array([1, 0.5])
print(f'Number of objects = {n} \
       \nLearning rate = {eta} \
       \nInitial weights = {W} \n')

for i in range(n_iter):
    y_pred = np.dot(X, W)
    err = calc_mse(y, y_pred)
#     for k in range(W.shape[0]):
#         W[k] -= eta * (1/n * 2 * X[:, k] @ (y_pred - y))
    # ИЗМЕНЕНИЯ
    W -= eta * (2/n * X.T @ (X @ W - y))
    # ИЗМЕНЕНИЯ
    #
    if i % 10 == 0:
        print(f'Iteration #{i}: W_new = {W}, MSE = {round(err,2)}')
```

```
Number of objects = 10
Learning rate = 0.01
Initial weights = [1.  0.5]

Iteration #0: W_new = [2.08 4.27], MSE = 3047.75
Iteration #10: W_new = [ 7.0011236 10.6169007], MSE = 738.65
Iteration #20: W_new = [10.3486292  10.10603105], MSE = 622.03
Iteration #30: W_new = [13.38789582  9.55618391], MSE = 525.24
Iteration #40: W_new = [16.16088505  9.05336203], MSE = 444.66
Iteration #50: W_new = [18.69110735  8.59454545], MSE = 377.58
Iteration #60: W_new = [20.99981865  8.17589626], MSE = 321.72
Iteration #70: W_new = [23.10641138  7.79389815], MSE = 275.22
Iteration #80: W_new = [25.02858024  7.44534246], MSE = 236.5
Iteration #90: W_new = [26.78247081  7.12730145], MSE = 204.27
Iteration #100: W_new = [28.38281518  6.83710367], MSE = 177.43
Iteration #110: W_new = [29.84305573  6.57231156], MSE = 155.08
Iteration #120: W_new = [31.17545797  6.33070096], MSE = 136.48
Iteration #130: W_new = [32.39121367  6.11024241], MSE = 120.99
Iteration #140: W_new = [33.50053475  5.90908413], MSE = 108.09
Iteration #150: W_new = [34.51273915  5.72553647], MSE = 97.36
Iteration #160: W_new = [35.43632906  5.55805768], MSE = 88.42
Iteration #170: W_new = [36.27906231  5.405241  ], MSE = 80.98
Iteration #180: W_new = [37.0480176   5.26580281], MSE = 74.78
Iteration #190: W_new = [37.74965389  5.13857189], MSE = 69.62
Iteration #200: W_new = [38.38986469  5.02247953], MSE = 65.33
Iteration #210: W_new = [38.97402756  4.9165506 ], MSE = 61.75
Iteration #220: W_new = [39.50704928  4.81989533], MSE = 58.77
Iteration #230: W_new = [39.99340705  4.73170185], MSE = 56.29
Iteration #240: W_new = [40.43718613  4.65122936], MSE = 54.23
Iteration #250: W_new = [40.84211409  4.57780191], MSE = 52.51
Iteration #260: W_new = [41.21159221  4.51080275], MSE = 51.08
Iteration #270: W_new = [41.54872398  4.4496691 ], MSE = 49.89
Iteration #280: W_new = [41.8563412   4.39388747], MSE = 48.9
Iteration #290: W_new = [42.13702774  4.34298929], MSE = 48.07
Iteration #300: W_new = [42.39314129  4.29654705], MSE = 47.39
Iteration #310: W_new = [42.6268331   4.25417064], MSE = 46.81
Iteration #320: W_new = [42.84006612  4.21550412], MSE = 46.34
Iteration #330: W_new = [43.03463143  4.1802227 ], MSE = 45.94
Iteration #340: W_new = [43.21216332  4.14803003], MSE = 45.61
Iteration #350: W_new = [43.37415299  4.1186557 ], MSE = 45.34
Iteration #360: W_new = [43.5219611   4.09185298], MSE = 45.11
```

```
Iteration #370: W_new = [43.6568292   4.06739673], MSE = 44.92
Iteration #380: W_new = [43.77989013  4.04508153], MSE = 44.76
Iteration #390: W_new = [43.89217756  4.02471993], MSE = 44.63
Iteration #400: W_new = [43.99463466  4.00614091], MSE = 44.52
Iteration #410: W_new = [44.08812206  3.98918842], MSE = 44.42
Iteration #420: W_new = [44.173425    3.97372004], MSE = 44.35
Iteration #430: W_new = [44.25126001  3.95960587], MSE = 44.28
Iteration #440: W_new = [44.32228086  3.94672733], MSE = 44.23
Iteration #450: W_new = [44.38708413  3.93497626], MSE = 44.19
Iteration #460: W_new = [44.44621412  3.92425394], MSE = 44.15
Iteration #470: W_new = [44.50016751  3.91447033], MSE = 44.12
Iteration #480: W_new = [44.5493975   3.90554323], MSE = 44.1
Iteration #490: W_new = [44.5943176   3.89739766], MSE = 44.07
Iteration #500: W_new = [44.63530512  3.8899652 ], MSE = 44.06
Iteration #510: W_new = [44.67270435  3.88318343], MSE = 44.04
Iteration #520: W_new = [44.70682942  3.87699538], MSE = 44.03
Iteration #530: W_new = [44.73796697  3.87134906], MSE = 44.02
Iteration #540: W_new = [44.76637856  3.86619706], MSE = 44.01
Iteration #550: W_new = [44.79230282  3.86149609], MSE = 44.0
Iteration #560: W_new = [44.81595752  3.85720668], MSE = 44.0
Iteration #570: W_new = [44.83754134  3.85329279], MSE = 43.99
Iteration #580: W_new = [44.85723558  3.84972154], MSE = 43.99
Iteration #590: W_new = [44.87520567  3.84646294], MSE = 43.99
Iteration #600: W_new = [44.89160255  3.84348962], MSE = 43.98
Iteration #610: W_new = [44.90656394  3.8407766 ], MSE = 43.98
Iteration #620: W_new = [44.92021553  3.83830109], MSE = 43.98
Iteration #630: W_new = [44.93267197  3.83604231], MSE = 43.98
Iteration #640: W_new = [44.94403791  3.83398127], MSE = 43.98
Iteration #650: W_new = [44.95440879  3.83210067], MSE = 43.97
```

3*. Вместо того, чтобы задавать количество итераций, задайте другое условие останова алгоритма - когда веса перестают изменяться меньше определенного порога ϵ.

Ввод [45]:

```
n = X.shape[0]

eta = 1e-2
n_iter = 6000

min_weight_dist = 1e-8

W = np.array([1, 0.5])
print(f'Number of objects = {n} \
       \nLearning rate = {eta} \
       \nInitial weights = {W} \n')

for i in range(n_iter):
    y_pred = np.dot(X, W)
    err = calc_mse(y, y_pred)
#     for k in range(W.shape[0]):
#         W[k] -= eta * (1/n * 2 * X[:, k] @ (y_pred - y))
    # ИЗМЕНЕНИЯ
    new_w = W - eta * (2/n * X.T @ (X @ W - y))
    weight_dist = np.linalg.norm(new_w - W, ord=2)
    # ИЗМЕНЕНИЯ
    #
    W = new_w
    if i % 10 == 0:
        print(f'Iteration #{i}: W_new = {W}, MSE = {round(err,2)}')

    if weight_dist <= min_weight_dist:
        print(f'Iteration #{i}: W_new = {W}, MSE = {round(err,2)}')
        break;
```

```
Number of objects = 10
Learning rate = 0.01
Initial weights = [1.  0.5]

Iteration #0: W_new = [2.08 4.27], MSE = 3047.75
Iteration #10: W_new = [ 7.0011236 10.6169007], MSE = 738.65
Iteration #20: W_new = [10.3486292  10.10603105], MSE = 622.03
Iteration #30: W_new = [13.38789582  9.55618391], MSE = 525.24
Iteration #40: W_new = [16.16088505  9.05336203], MSE = 444.66
Iteration #50: W_new = [18.69110735  8.59454545], MSE = 377.58
Iteration #60: W_new = [20.99981865  8.17589626], MSE = 321.72
Iteration #70: W_new = [23.10641138  7.79389815], MSE = 275.22
Iteration #80: W_new = [25.02858024  7.44534246], MSE = 236.5
Iteration #90: W_new = [26.78247081  7.12730145], MSE = 204.27
Iteration #100: W_new = [28.38281518  6.83710367], MSE = 177.43
Iteration #110: W_new = [29.84305573  6.57231156], MSE = 155.08
Iteration #120: W_new = [31.17545797  6.33070096], MSE = 136.48
Iteration #130: W_new = [32.39121367  6.11024241], MSE = 120.99
Iteration #140: W_new = [33.50053475  5.90908413], MSE = 108.09
```

Дополнительные материалы

1. Обучение с учителем. Обучение без учителя. Обучение с подкреплением. Видео (https://youtu.be/ku2oPMPht2I)
2. Вывод аналитической формулы решения уравнения линейной регрессии (https://habr.com/ru/company/ods/blog/323890/#metod-naimenshih-kvadratov) (см. пункт 1.2)
3. Математическое описание метода градиентного спуска (http://www.machinelearning.ru/wiki/index.php?title=%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B3%D1%80%D0%B0%D0%B4%D0%B8%D0%B5%D0%BD%D1%82%D0%BD
4. Документация NumPy (https://docs.scipy.org/doc/numpy-1.16.0/reference/routines.html)
5. Метрики для задачи регрессии. Видео (https://youtu.be/vh2smjQyhp8)

Summary

- Линейная регрессия - простой, но зачастую эффективный, способ приближать вещественную целевую переменную через линейную комбинацию признаков

- Решение регрессии - МНК, можно решать аналитически, но на практике - градиентный спуск (GD, Gradient Descent)
- MSE удобна для градиентного спуска, так как дифференцируема
- Максимальное качество градиентного спуска достигается малыми шагами и большим кол-вом итераций

Определения

Машинное обучение

Машинное обучение — дисциплина, заключающаяся в извлечении знаний из известных данных.

Признак — это индивидуальное измеримое свойство или характеристика наблюдения.

Обучающая выборка — набор структурированных данных, используемый для обучения моделей.

Обучение с учителем — это направление машинного обучения, объединяющее алгоритмы и методы построения моделей на основе множества примеров, содержащих пары "известный вход - известный выход".

Обучение без учителя — направление машинного обучения, в которой для коррекции параметров обучаемой модели не используется целевая функция. Иными словами, в обучающих примерах при обучении без учителя не нужно иметь заранее заданные выходы модели.

Обучение с подкреплением — раздел машинного обучения, изучающий поведение интеллектуальных агентов, действующих в некоторой среде и принимающих решения.

Линейные модели

Линейные модели — такие модели, которые сводятся к суммированию значений признаков с некоторыми весами

Линейная регрессия — модель зависимости переменной x от одной или нескольких других переменных (факторов, регрессоров, независимых переменных) с линейной функцией зависимости.

Метод наименьших квадратов — способ вычисления весов линейной модели путем минимизации среднеквадратичного отклонения.

Градиентный спуск

Градиент функции — n-мерный вектор из частных производных. Задает направление наискорейшего роста функции.

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

Градиентный спуск — метод нахождения локального экстремума функции (минимума или максимума) с помощью движения вдоль