

## Содержание

<b>1</b>	<b>Лекция 1</b>	<b>2</b>
1.1	Как устроен курс . . . . .	2
1.2	Практика . . . . .	2
1.3	SymPy . . . . .	2
1.4	Рисуем графики . . . . .	2
1.4.1	Подключаем библиотеку . . . . .	3
1.4.2	Рисуем график функции $f(x)$ . . . . .	3
1.4.3	Изменяем размер картинки и граничные значения по осям . . . . .	3
1.4.4	Добавим названия осей и подпись к графику . . . . .	3
1.4.5	Добавим оси и подпишем под картинкой информацию об экстремумах и точках перегиба . . . . .	3
1.5	Метод градиентного спуска . . . . .	4
1.5.1	Теорема о градиенте . . . . .	4
1.5.2	Применение в машинном обучении . . . . .	4
1.5.3	Идея применения градиентного спуска . . . . .	4
1.5.4	Метод градиентного спуска на пальцах . . . . .	4
1.5.5	Метод градиентного спуска (одномерный случай) . . . . .	4
1.5.6	Метод градиентного спуска (общий случай) . . . . .	4
1.5.7	Параметр learning rate . . . . .	4
1.5.8	Теорема о поиске в выпуклой гладкой функции . . . . .	5
1.6	Реализация градиентного спуска на python . . . . .	5
<b>2</b>	<b>Лекция 2</b>	<b>6</b>
2.1	Линейная регрессия . . . . .	6
2.2	Функция ошибок MSE . . . . .	6
2.3	Матричное представление MSE . . . . .	6
2.4	Как решать задачу минимизации MSE . . . . .	6
2.4.1	Аналитическое решение . . . . .	6
2.4.2	Приближённое решение . . . . .	6
2.5	Производная по вектору . . . . .	6
2.5.1	Пример 1. Подсчёт градиента скалярного произведения . . . . .	6
2.5.2	Пример 2. Подсчёт градиента от матрицы . . . . .	7
2.6	Минимизация MSE с помощью градиента . . . . .	7
2.7	Градиент функции потерь . . . . .	7
2.8	Реализация на питоне . . . . .	7
2.8.1	Генерирование данных для задачи регрессии . . . . .	7
2.8.2	Функция подсчёта ошибки . . . . .	7
2.8.3	Реализация градиентного спуска . . . . .	8
2.8.4	Функция предсказания модели . . . . .	8
2.8.5	Применение градиентного спуска . . . . .	8

# 1. Лекция 1

## 1.1. Как устроен курс

- 1 модуль - матан и линал
- 2 модуль - дискра, тервер
- 3/4 модуль - статистика

## 1.2. Практика

Исследовать функцию

$$f(x) = x^3 - 3x^2 + 4$$

Найдём

## 1.3. SymPy

В Питоне есть библиотека SymPy, которая предоставляет интерфейс для вычисления производных

```
!pip install sympy
```

Далее в питоне зададим переменную и производную:

```
import sympy as sp
x = sp.Symbol('x')
sp.diff(x**6)
```

Теперь будем анализировать функцию из практики:

```
def f(x):
    return x**3 - 3*x**2 + 4
```

Чтобы найти нули функции, надо решить уравнение  $f(x) = 0$ . В SymPy для этого есть функция solve:

```
sp.solve(f(x), x)
[-1, 2]
```

Теперь найдём производную функции  $f(x)$  и затем её нули, чтобы найти экстремумы

```
df_x = sp.diff(f(x))
#df_x == 3x^2
```

```
sp.solve(df_x, x)
# [0, 2]
```

```
f(0), f(2)
# (4, 0)
```

Точно также очень просто можем находить вторую производную и находить точки перегиба функции

```
d2f_x = sp.diff(df_x)
d2f_x
# 6x - 6
```

```
sp.solve(d2f_x, x)
#[1]
```

```
f(1)
#2
```

## 1.4. Рисуем графики

Что нам нужно будет сделать?

- Нарисовать график  $f(x)$ , подписать оси
- Напечатать под графиком при помощи *Markdown* экстремумы, точки перегиба и значения функции  $f(x)$  в этих точках

#### 1.4.1. Подключаем библиотеку

```
import matplotlib.pyplot as plt
%matplotlib inline # —
```

#### 1.4.2. Рисуем график функции $f(x)$

```
import numpy as np

x_values = [x for x in np.arange(-5, 5, 0.1)]
# or      = np.linspace(-5, 5, 100)
f_values = [f(x) for x in x_values]

plt.plot(x_values, f_values)
```

**Вставить график**

#### 1.4.3. Изменяем размер картинки и граничные значения по осям

```
plt.figure(figsize=(10, 10))

plt.plot(x_values, y_values)

plt.xlim([-3, 5])
plt.ylim([-5, 7])
```

**Вставить график**

#### 1.4.4. Добавим названия осей и подпись к графику

```
plt.title('Graph_of_fucntion_f(x)_with_extremum_and_dots_of...')

plt.xlabel('x')
plt.ylabel('f(x)')
```

**Вставить график**

#### 1.4.5. Добавим оси и подпишем под картинкой информацию об экстремумах и точках перегиба

```
import numpy as np

x_values = [x for x in np.arange(-5, 5, 0.1)]
f_values = [f(x) for x in x_values]

plt.figure(figsize=(10,10))

plt.axvline(x=0, c = 'black')
plt.axhline(y=0, c = 'black')

plt.plot(x_values, f_values)

plt.xlim([-3, 5])
plt.ylim([-5, 7])

plt.title('Graph_of_fucntion_f(x)_with_extremum_and_dots_of...')

plt.xlabel('x')
plt.ylabel('f(x)')

plt.show()
```

**Вставить график**

## 1.5. Метод градиентного спуска

### 1.5.1. Теорема о градиенте

**Градиент** - это вектор, в направлении которого функция растёт быстрее всего.

**Антиградиент** (вектор противоположный градиенту) - вектор, в направлении которого функция быстрее всего убывает.

### 1.5.2. Применение в машинном обучении

Для чего нам это нужно? В машинном обучении мы минимизируем значение функции, которая показывает ошибку модели. Иными словами: наша задача при обучении модели - найти такие веса  $\mathbf{w}$ , на которых достигается **минимум функции ошибок**.

В простейшем случае, если ошибка среднеквадратическая, то её график - парабола.

### 1.5.3. Идея применения градиентного спуска

На каждом шаге (на каждой итерации метода) движемся в сторону антиградиента функции потерь!

То есть на каждом шаге движемся в направлении уменьшения ошибки.

Вектор градиента функции потерь обозначают **grad Q** или  $\nabla Q$

### 1.5.4. Метод градиентного спуска на пальцах

- Встаём в некоторую точку функции
- Вычисляем градиент
- Переходим в новую точку в направлении антиградиента
- Повторяем процесс из новой точки

### 1.5.5. Метод градиентного спуска (одномерный случай)

Пусть у нас только один вес -  $w$ .

Тогда при добавлении к весу  $w$  слагаемого  $-\frac{\partial Q}{\partial w}$  функция  $Q(w)$  убывает.

Тогда алгоритм выглядит следующим образом:

- Инициализируем вес  $w^{(0)}$
- На каждом следующем шаге обновляем вес, добавляя  $-\frac{\partial Q}{\partial w}(w^{(k-1)})$ :

$$w^{(k)} = w^{(k-1)} - \frac{\partial Q}{\partial w}(w^{(k-1)})$$

### 1.5.6. Метод градиентного спуска (общий случай)

Пусть  $w_0, w_1, \dots, w_n$  - веса, которые мы ищем.

Тогда  $\nabla Q(w) = \left\{ \frac{\partial Q}{\partial w_0}, \frac{\partial Q}{\partial w_1}, \dots, \frac{\partial Q}{\partial w_n} \right\}$

Тогда алгоритм выглядит так:

- Инициализируем веса  $w^{(0)}$  (заметим, что это вектор весов)
- На каждом шаге обновляем веса по формуле:

$$w^{(k)} = w^{(k-1)} - \nabla Q(w^{(k-1)})$$

### 1.5.7. Параметр learning rate

В формулу обычно добавляют параметр  $\eta$  - величина градиентного спуска (**learning rate**). Он отвечает за скорость движения в сторону антиградиента:

- Инициализируем веса  $w^{(0)}$  (заметим, что это вектор весов)
- На каждом шаге обновляем веса по формуле:

$$w^{(k)} = w^{(k-1)} - \eta \nabla Q(w^{(k-1)})$$

### 1.5.8. Теорема о поиске в выпуклой гладкой функции

Если функция  $Q(w)$  выпуклая и гладкая, а также имеет минимум в точке  $w^*$ , то метод градиентного спуска при аккуратно подобранному  $\eta$  через некоторое число шагов гарантированно попадает в малую окрестность точки  $w^*$ .

## 1.6. Реализация градиентного спуска на python

```
def gradient_descent(x_start, learning_rate, epsilon, num_iterations):
    x_curr = x_start
    df_x = sp.diff(f(x))

    trace = []
    trace.append(x_curr)

    for i in range(num_iterations):
        x_new = x_curr + df_x.subs(x, x_curr)
        trace.append(x_new)

        if abs(x_new - x_curr) < epsilon:
            return x_curr, trace

    return x_curr, trace
```

## 2. Лекция 2

### 2.1. Линейная регрессия

**Линейная регрессия** - функция  $a(x) = \omega_0 + \omega_1 x_1 + \omega_2 x_2 + \dots + \omega_l x_l$ , где  $x$  - **вектор признаков**

Также есть **целевая переменная**, которую мы предсказываем -  $y$

$w$  - **веса** линейной регрессии

Запишем в другой форме:  $a(x) = \omega_0 + \sum_{i=1}^l \omega_i x_i$

Также мы это можем записать в другой форме: давайте добавим ещё один признак у всех объектов, который будет равен единице:  $x = (1, x_1, x_2, \dots, x_n)$ , и тогда всё записывается ещё красивее:

$$a(x) = \sum_{i=0}^l \omega_i x_i = (\vec{\omega}, \vec{x})$$

Где  $a(x) = (\vec{\omega}, \vec{x})$  - предсказание модели на объекте  $x$ . Но это предсказание для одного объекта.

Мы можем записать предсказания в матричном виде для нескольких объектов. Возьмём  $X$  - матрицу объект-признак. В каждой строке описан один объект, а кол-во строк - это кол-во объектов.

В матричном виде предсказание выглядит как  $a(X) = X \cdot w$

### 2.2. Функция ошибок MSE

**MSE** (Mean Squared Error) =  $\frac{1}{d} \sum_{i=1}^d (a(x_i) - y_i)^2$ , где  $d$  - количество данных. При этом в задаче обучения мы хотим  $MSE \rightarrow \min_{\vec{\omega}}$ . Мы уже научились и можем делать минимизацию функции с помощью Градиентного спуска.

### 2.3. Матричное представление MSE

$$MSE = \frac{1}{d} \|X\omega - y\|^2$$

Где  $\|\vec{a}\| = \sqrt{a_1^2 + a_2^2 + \dots + a_n^2}$

Соответственно  $\|\vec{a}\|^2 = a_1^2 + a_2^2 + \dots + a_n^2$

### 2.4. Как решать задачу минимизации MSE

#### 2.4.1. Аналитическое решение

Это решение, которое даёт точное решение

Решаем уравнение  $\nabla_{\omega} Q(\omega) = 0$

#### 2.4.2. Приближённое решение

С помощью GD шагаем  $\omega = \omega - \eta \nabla_{\omega} Q(\omega)$

### 2.5. Производная по вектору

Пусть у нас есть  $\vec{x} = (x_1, \dots, x_n)$

Градиент функции  $f(x)$  рассчитывается как  $\nabla_x f(x) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n})$

Если мы хотим взять градиент по матрице  $A$  от функции  $f$ , то это выглядит как  $\nabla_A f(A) = \begin{pmatrix} \frac{\partial f}{\partial A_{11}} & \dots & \frac{\partial f}{\partial A_{1n}} \\ \dots & \dots & \dots \\ \frac{\partial f}{\partial A_{n1}} & \dots & \frac{\partial f}{\partial A_{nn}} \end{pmatrix}$

#### 2.5.1. Пример 1. Подсчёт градиента скалярного произведения

Пусть у нас есть вектор весов  $\vec{\omega}$  и вектор  $\vec{x}$ . Есть скалярное произведение  $(\vec{\omega}, \vec{x})$ . Мы хотим посчитать  $\nabla_x (\vec{\omega}, \vec{x})$ .

Мы знаем, что  $\frac{\partial}{\partial x_i} (\vec{\omega}, \vec{x}) = \frac{\partial}{\partial x_i} (\omega_1 x_1 + \dots + \omega_n x_n) = \omega_i$

Тогда  $\nabla_x (\vec{\omega}, \vec{x}) = (\frac{\partial}{\partial x_1} (\vec{\omega}, \vec{x}), \frac{\partial}{\partial x_2} (\vec{\omega}, \vec{x}), \dots, \frac{\partial}{\partial x_n} (\vec{\omega}, \vec{x})) = (\omega_1, \omega_2, \dots, \omega_n) = \vec{\omega}$

### 2.5.2. Пример 2. Подсчёт градиента от матрицы

Пусть есть матрица  $A_{n \times n}$  и вектор  $\vec{x} \in \mathbb{R}^n$ .

Функция  $x^T A x$  - это число (давайте посмотрим на размерности)

$$(1 \times n)(n \times n)(n \times 1) = (1 \times n)(n \times 1) = (1 \times 1)$$

Теперь мы хотим от этой функции находить градиент:

$$\begin{aligned} \text{Давайте попробуем посчитать } \frac{\partial}{\partial x_i} x^T A x &= \frac{\partial}{\partial x} \sum_{j=1}^n x_j (A x)_j = \frac{\partial}{\partial x_i} \sum_{j=1}^n x_j \left( \sum_{k=1}^n a_{jk} x_k \right) = \frac{\partial}{\partial x_i} \sum_{j=1}^n \sum_{k=1}^n a_{jk} x_j x_k = \\ &= \sum_{j=1, j \neq i}^n a_{ji} x_j + \sum_{k=1, k \neq i}^n a_{ik} x_i + 2a_{ii} x_i = \sum_{i=1}^n \sum_{j=1}^n (a_{ij} + a_{ji}) x_j - i\text{-я производная.} \end{aligned}$$

$$\text{Тогда } \nabla_x (x^T A x) = (A + A^T)x$$

## 2.6. Минимизация MSE с помощью градиента

Вспоминаем, что MSE выглядит как  $\|y - X\omega\|^2$ . Но это можно переписать в явном виде без квадрата:

$$\|y - X\omega\|^2 = (y - X\omega)^T (y - X\omega) \rightarrow \min_{\omega}$$

Раскрываем скобки для поиска градиента:  $\nabla_{\omega} ((y^T y)^0 - \omega^T X^T y - y^T X \omega + \omega^T X^T X \omega) = 0$

$\nabla_{\omega} (-\omega^T X^T y - y^T X \omega + \omega^T X^T X \omega) = -X^T y - X^T y + 2X^T X \omega = 0$  (для последнего слагаемого смотрим вывод пункта 2.5.2)

Перекинем слагаемые в разные стороны:  $2X^T X \omega = 2X^T y$ . Сократим на двойку. Мы бы могли сократить матрицы, но обратной может не быть. Зато мы можем с каждой из сторон умножить на обратную матрицу к  $X^T X$ :

$$X^T X \omega = X^T y \rightarrow (X^T X)^{-1} (X^T X) \omega = (X^T X)^{-1} X^T y \rightarrow \omega = (X^T X)^{-1} X^T y$$

## 2.7. Градиент функции потерь

Из подсчитанного можем сказать, что градиент функции потерь для MSE будет выглядеть как

$$\nabla Q(\omega) = 2X^T (X\omega - y)$$

Но давайте будем находить  $\omega$  с помощью GD:

- На шаге обновления точки, у нас  $\omega_{next} = \omega_{prev} - \eta \nabla Q(\omega)$
- Запишем зная, чему равно  $\nabla Q(\omega)$ :  $\omega_{next} = \omega_{prev} - 2\eta X^T (X\omega_{prev} - y)$

## 2.8. Реализация на питоне

### 2.8.1. Генерирование данных для задачи регрессии

Давайте сгенерируем данные и визуализируем их:

```
import random
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

X = np.linspace(-10, 10, 100)

print(X.shape)

y = X * (np.random.random_sample(len(X)) + 0.5)
X = X.reshape(len(X), 1)

print(X.shape)

plt.scatter(X, y)
```

### 2.8.2. Функция подсчёта ошибки

Также давайте напишем свой MSE:

```
def MSE(X, y, theta):
    m = len(y)

    error = (1./m) * (np.linalg.norm(X @ theta - y) ** 2)
    return error
```

### 2.8.3. Реализация градиентного спуска

Теперь у нас есть всё, чтобы реализовать свой градиентный спуск:

```
def gradient_descent(X, y, learning_rate, iterations):

    X = np.hstack((np.ones((X.shape[0], 1)), X)) # add column of ones
    params = np.random.rand(X.shape[1])

    m = X.shape[0]

    cost_track = np.zeros((iterations, 1))

    for i in range(iterations):
        params = params - 2./m * learning_rate * (X.T @ ((X @ params) - y))
        cost_track[i] = MSE(X, y, params)

    return cost_track, params
```

### 2.8.4. Функция предсказания модели

Записать предсказание модели можно очень просто:

```
def predict(X, params):
    X = np.hstack((np.ones((X.shape[0], 1)), X))
    return X @ params
```

### 2.8.5. Применение градиентного спуска

Применяем градиентный спуск:

```
track, weights = stochastic_gradient_descent(X, y, 0.01, 100)
plt.plot(track) # visualize errors
```

Теперь сделаем предсказание и посмотрим на визуализацию этого предсказания:

```
pred = predict(X, weights)
plt.scatter(X, y)
plt.plot(X, pred, '-', c = 'r')
```