Семинар: Градиентный спуск. Задачи

B [37]:
```
from typing import Iterable, List

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import abc
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

%matplotlib inline
```

Часть 1. Градиентный спуск (вспомним формулы)

Функционал ошибки, который мы применяем в задаче регрессии — Mean Squared Error:

$$Q(w, X, y) = \frac{1}{\ell}\sum_{i=1}^{\ell}(\langle x_i, w\rangle - y_i)^2$$

где x_i — это i-ый объект датасета, y_i — правильный ответ для i-го объекта, а w — веса нашей линейной мод

Можно показать, что для линейной модели, функционал ошибки можно записать в матричном виде следующи

$$Q(w, X, y) = \frac{1}{l}(y - Xw)^T(y - Xw)$$

или

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

где X — это матрица объекты-признаки, а y — вектор правильных ответов

Для того чтобы воспользоваться методом градиентного спуска, нам нужно посчитать градиент нашего функци выглядеть так:

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

Формула для одной итерации градиентного спуска выглядит следующим образом:

$$w^t = w^{t-1} - \eta\nabla_w Q(w^{t-1}, X, y)$$

Где w^t — значение вектора весов на t-ой итерации, а η — параметр learning rate, отвечающий за размер шаг

Часть 2. Линейная регрессия (продолжение части 1 - используют части 1).

Создадим класс для линейной регрессии. Он будет использовать интерфейс, знакомый нам из библиотеки `sk`

В методе `fit` мы будем подбирать веса `w` при помощи градиентного спуска нашим методом `gradient_des`

В методе `predict` мы будем применять нашу регрессию к датасету,

Задание 2.1: Допишите код в методах `fit` и `predict` класса `LinearRegression_1`

В методе `fit` вам нужно инициализировать веса `w` (например, из члучайного распределения), применить н сохранить последние веса `w` из траектории.

В методе `predict` вам нужно применить линейную регрессию и вернуть вектор ответов.

Обратите внимание, что объект лосса (функционала ошибки) передаётся в момент инициализации и хранитс использовать в `fit` для `gradient_descent` (например, с `n_iterations: int = 1000`).

In [38]:
```
class BaseLoss(abc.ABC):
    @abc.abstractmethod
    def calc_loss(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) -> float:
        raise NotImplementedError

    @abc.abstractmethod
    def calc_grad(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) -> np.ndarray:
        raise NotImplementedError

class MSELoss(BaseLoss):
    def calc_loss(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) -> float:
        return (np.linalg.norm(np.matmul(X,w) - y)**2)/len(y)

    def calc_grad(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) -> np.ndarray:
        return 2/len(y)*np.matmul(np.transpose(X), (np.matmul(X, w) - y))

```

In [39]:
```
def gradient_descent(
    w_init: np.ndarray,
    X: np.ndarray,
    y: np.ndarray,
    loss: BaseLoss,
    lr: float,
    n_iterations: int = 1000,
    ) -> List[np.ndarray]:
        grad_list = []
        for i in range(n_iterations):
            grad_list.append(w_init)
            w_init = w_init - lr * loss.calc_grad(X, y, w_init)
        return grad_list
```

In [40]:
```
class LinearRegression_1:
    def __init__(self, loss: BaseLoss, lr: float = 0.01) -> None:
        self.loss = loss #функционал ошибки
        self.lr = lr #градиентный шаг

    def fit(self, X: np.ndarray, y: np.ndarray) -> "LinearRegression_1":
        X = np.asarray(X)
        y = np.asarray(y)
        X = np.hstack([X, np.ones([X.shape[0], 1])])
        w_init = np.random.normal(size=(X.shape[1],))

        w_arr = gradient_descent(w_init, X, y, self.loss, self.lr, 1000)
        self.w = w_arr[-1]

        return self


    def predict(self, X: np.ndarray) -> np.ndarray:
        assert hasattr(self, "w"), "Linear regression must be fitted first" #проверяем, о
        X = np.asarray(X)
        X = np.hstack([X, np.ones([X.shape[0], 1])])

        y = []
        for i in range(X.shape[0]):
            y1=np.sum(X[i]*self.w)
            y.append(y1)

        return y
```

Класс линейной регрессии создан. Более того, мы можем управлять тем, какой функционал ошибки мы оптим
разные классы в параметр loss при инициализации.

Будем применять нашу регрессию на реальном датафрейме. Загрузим датафрейм с машинами (см. семинар

In [41]:
```
X_raw = pd.read_csv(
    "http://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data",
    header=None,
    na_values=["?"],
)

X_raw.head()
X_raw = X_raw[~X_raw[25].isna()].reset_index(drop=True)
```

In [42]:
```
y = X_raw[25]
X_raw = X_raw.drop(25, axis=1)
```

Задание 2.2: Обработайте датасет нужными методами, чтобы на нём можно было обучать линейную регресс
sklearn-knn.ipynb):

- Заполните пропуски средними (библиотека SimpleImputer)
- Переведите категориальные признаки в числовые (в методе get_dummies использовать drop_first=True.)

- Разделите датасет на обучающую и тестовую выборку (задать: доля тестовой выборки равна 0.3, random shuffle=True)
- Нормализуйте числовые признаки (при помощи бибилиотеки StandardScaler)

In [43]:
```
X_raw.isna().any()  # сеачала проверим есть ли пропуски вообще
```

Out[43]:
```
0     False
1      True
2     False
3     False
4     False
5      True
6     False
7     False
8     False
9     False
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18     True
19     True
20    False
21     True
22     True
23    False
24    False
dtype: bool
```

In [44]:
```
cat_features_mask = (X_raw.dtypes == "object").values    # делаем маску для определения с
X_real = X_raw[X_raw.columns[~cat_features_mask]]        # отбираем столбцы с числовыми д
```

In [45]:
```
mis_replacer = SimpleImputer(strategy="mean")            # заводим переменную реплэйсера
data = mis_replacer.fit_transform(X_real)                # заполняем пропуски
X_no_mis_real = pd.DataFrame(data, columns=X_real.columns)    # возвращаем все к виду дат
```

In [46]:
```
X_cat = X_raw[X_raw.columns[cat_features_mask]].fillna("")      # в остальных столбцах вм
X_no_mis = pd.concat([X_no_mis_real, X_cat],axis=1)             # соединяем таблички
X_no_mis.head(10)
```

Out[46]:

	0	1	9	10	11	12	13	16	18	19	...	2	3	4	5	6	7	
0	3.0	122.0	88.6	168.8	64.1	48.8	2548.0	130.0	3.47	2.68	...	alfa-romero	gas	std	two	convertible	rwd	f
1	3.0	122.0	88.6	168.8	64.1	48.8	2548.0	130.0	3.47	2.68	...	alfa-romero	gas	std	two	convertible	rwd	f
2	1.0	122.0	94.5	171.2	65.5	52.4	2823.0	152.0	2.68	3.47	...	alfa-romero	gas	std	two	hatchback	rwd	f
3	2.0	164.0	99.8	176.6	66.2	54.3	2337.0	109.0	3.19	3.40	...	audi	gas	std	four	sedan	fwd	f
4	2.0	164.0	99.4	176.6	66.4	54.3	2824.0	136.0	3.19	3.40	...	audi	gas	std	four	sedan	4wd	f
5	2.0	122.0	99.8	177.3	66.3	53.1	2507.0	136.0	3.19	3.40	...	audi	gas	std	two	sedan	fwd	f
6	1.0	158.0	105.8	192.7	71.4	55.7	2844.0	136.0	3.19	3.40	...	audi	gas	std	four	sedan	fwd	f
7	1.0	122.0	105.8	192.7	71.4	55.7	2954.0	136.0	3.19	3.40	...	audi	gas	std	four	wagon	fwd	f
8	1.0	158.0	105.8	192.7	71.4	55.9	3086.0	131.0	3.13	3.40	...	audi	gas	turbo	four	sedan	fwd	f
9	2.0	192.0	101.2	176.8	64.8	54.3	2395.0	108.0	3.50	2.80	...	bmw	gas	std	two	sedan	rwd	f

10 rows × 25 columns

B [47]:
```
X_dum = pd.get_dummies(X_no_mis, drop_first=True)  # кодируем категориальные признаки one
X_dum.head(10)
```

Out[47]:

	0	1	9	10	11	12	13	16	18	19	...	15_three	15_twelve	15_two	17_2bbl	17_4bbl
0	3.0	122.0	88.6	168.8	64.1	48.8	2548.0	130.0	3.47	2.68	...	0	0	0	0	0
1	3.0	122.0	88.6	168.8	64.1	48.8	2548.0	130.0	3.47	2.68	...	0	0	0	0	0
2	1.0	122.0	94.5	171.2	65.5	52.4	2823.0	152.0	2.68	3.47	...	0	0	0	0	0
3	2.0	164.0	99.8	176.6	66.2	54.3	2337.0	109.0	3.19	3.40	...	0	0	0	0	0
4	2.0	164.0	99.4	176.6	66.4	54.3	2824.0	136.0	3.19	3.40	...	0	0	0	0	0
5	2.0	122.0	99.8	177.3	66.3	53.1	2507.0	136.0	3.19	3.40	...	0	0	0	0	0
6	1.0	158.0	105.8	192.7	71.4	55.7	2844.0	136.0	3.19	3.40	...	0	0	0	0	0
7	1.0	122.0	105.8	192.7	71.4	55.7	2954.0	136.0	3.19	3.40	...	0	0	0	0	0
8	1.0	158.0	105.8	192.7	71.4	55.9	3086.0	131.0	3.13	3.40	...	0	0	0	0	0
9	2.0	192.0	101.2	176.8	64.8	54.3	2395.0	108.0	3.50	2.80	...	0	0	0	0	0

10 rows × 65 columns

B [48]:
```
# при нормализации ошибку выдало, что в столбцах есть числовые названия, так что нужно пе
X_dum.columns = [str(i) for i in X_dum.columns]
```

B [49]:
```
X_train, X_test, y_train, y_test = train_test_split(X_dum, y, random_state=42, test_size=
```

B [50]:
```
scaler = StandardScaler()                    # создаем scaler
scaler.fit(X_train)                          # обучаем его
scaled_X_train = scaler.transform(X_train)   # нормализуем X_train
scaled_X_test = scaler.transform(X_test)     # нормализуем X_test
```

Задание 2.3: Обучите написанную вами линейную регрессию на обучающей выборке

Создаем объект линейной регрессии для `MSELoss` :

B [51]:
```
lr_1 = LinearRegression_1(MSELoss()) #создаем регрессор
```

B [52]:
```
lr_1.fit(scaled_X_train, y_train)    # обучаем модель
y_pred = lr_1.predict(scaled_X_test) # делаем предикты
```

Задание 2.4: Посчитайте ошибку обученной регрессии на обучающей и тестовой выборке при помощи метод `r2_score` из `sklearn.metrics` .

B [53]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

B [54]:
```
print(f"MSE: {mean_squared_error(y_test, y_pred)**0.5}")        #смотрим среднеквадратичн

print('Коэффициент детерминации: ', r2_score(y_test, y_pred))
```

```
MSE: 2517.9510100373222
Коэффициент детерминации:  0.9331156622103828
```

Добавим к модели L2 регуляризацию (для борьбы с переобучением). Для этого нам нужно создать новый кла ошибки и его градиента.

Формула функционала ошибки для MSE с L2 регуляризацией выглядит так:

$$Q(w, X, y) = \frac{1}{\ell}\sum_{i=1}^{\ell}(\langle x_i, w\rangle - y_i)^2 + \lambda||w||^2$$

Или в матричном виде:

$$Q(w, X, y) = \frac{1}{\ell}||Xw - y||^2 + \lambda||w||^2,$$

где λ — коэффициент регуляризации

Заметим, что (удобно для вычислений):

$$Q(w, X, y) = \frac{1}{\ell}||Xw - y||^2 + \lambda||w||^2 = \frac{1}{l}(y - Xw)^T(y - Xw) + \lambda w^T w.$$

Градиент $\nabla_w Q(w, X, y)$ выглядит так:

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

Задание 2.5: Реализуйте класс `MSEL2Loss`

Он должен вычислять значение функционала ошибки (лосс) $Q(w, X, y)$ и его градиент $\nabla_w Q(w, X, y)$ по фо

Подсказка: обратите внимание, что последний элемент вектора `w` — это bias (сдвиг) (в классе `LinearRegre` добавляется колонка из единиц — константный признак). bias регуляризовать не нужно. Поэтому не забудьте из `w` при подсчёте слагаемого $\lambda||w||^2$ в `calc_loss` и занулить его при подсчёте слагаемого $2\lambda w$ в `calc_g`

В [78]:
```
class MSEL2Loss(BaseLoss):
    def __init__(self, coef: float = 1.0):
        self.coef = coef

    def calc_loss(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) -> float:
        return (np.linalg.norm(np.matmul(X,w)-y)**2)/len(y) + self.coef*(np.linalg.norm(w

    def calc_grad(self, X: np.ndarray, y: np.ndarray, w: np.ndarray) -> np.ndarray:
        w_new = np.append(w[:-1],[0])
        return 2*np.matmul(np.transpose(X),(np.matmul(X,w)-y))/len(y) + 2*self.coef*w_new
```

В [79]:
```
# проверка:

# создадим объект лосса
losst = MSEL2Loss()

# создадим датасет
Xt = np.arange(200).reshape(20, 10)
yt = np.arange(20)

# создадим вектор весов
wt = np.arange(10)

#print(Xt)
#print(yt)
#print(wt)

# выведем значение лосса и градиента на этом датасете с этим вектором весов
print(losst.calc_loss(Xt, yt, wt))
print(losst.calc_grad(Xt, yt, wt))

# проверка, что методы реализованы правильно
assert losst.calc_loss(Xt, yt, wt) == 27410487.5, "Метод calc_loss реализован неверно"
assert np.allclose(
    losst.calc_grad(Xt, yt, wt),
    np.array(
        [
            1163180.0,
            1172283.0,
            1181386.0,
            1190489.0,
            1199592.0,
            1208695.0,
            1217798.0,
            1226901.0,
            1236004.0,
            1245089.0,
        ]
    ),
), "Метод calc_grad реализован неверно"

print("Всё верно!")
```

```
27410487.5
[1163180. 1172283. 1181386. 1190489. 1199592. 1208695. 1217798. 1226901.
 1236004. 1245089.]
Всё верно!
```

Теперь мы можем использовать лосс с l2 регуляризацией в нашей регрессии. Пусть:

В [57]:
```
lr_1_l2 = LinearRegression_1(MSEL2Loss(0.1))
```

Задание 2.6: Обучите регрессию с лоссом `MSEL2Loss` . Попробуйте использовать другие коэффициенты рег улучшить разультат на тестовой выборке? Сравните результат применения регрессии с регуляризацией к обу выборкам с результатом применения регрессии без регуляризации к тем же выборкам.(Для оценки качества и `mean_squared_error` и `r2_score` из `sklearn.metrics`).

В [72]:
```
lr_1_l2.fit(scaled_X_train, y_train)
y_predict = lr_1_l2.predict(scaled_X_test)
y_predict_train = lr_1_l2.predict(scaled_X_train)
print(f"MSE on Test: {mean_squared_error(y_test, y_predict)**0.5}")
print(f"MSE on Train: {mean_squared_error(y_train, y_predict_train)**0.5}")
print('Коэффициент детерминации на тестовой выборке: ', r2_score(y_test, y_predict))
print('Коэффициент детерминации на тренировочной выборке', r2_score(y_train, y_predict_tr
```

```
MSE on Test: 2518.250194286107
MSE on Train: 1393.7382610784412
Коэффициент детерминации на тестовой выборке:  0.9330997668024732
Коэффициент детерминации на тренировочной выборке 0.9596828392692338
```

В [73]:
```
lr_1_l2 = LinearRegression_1(MSEL2Loss(0.01))
lr_1_l2.fit(scaled_X_train, y_train)
y_predict = lr_1_l2.predict(scaled_X_test)
y_predict_train = lr_1_l2.predict(scaled_X_train)
print(f"MSE on Test: {mean_squared_error(y_test, y_predict)**0.5}")
print(f"MSE on Train: {mean_squared_error(y_train, y_predict_train)**0.5}")
print('Коэффициент детерминации на тестовой выборке: ', r2_score(y_test, y_predict))
print('Коэффициент детерминации на тренировочной выборке', r2_score(y_train, y_predict_tr
```

```
MSE on Test: 2525.426536883011
MSE on Train: 1408.5899787098672
Коэффициент детерминации на тестовой выборке:  0.9327179278005473
Коэффициент детерминации на тренировочной выборке 0.9588190194146218
```

В [74]:
```
lr_1_l2 = LinearRegression_1(MSEL2Loss(0.0001))
lr_1_l2.fit(scaled_X_train, y_train)
y_predict = lr_1_l2.predict(scaled_X_test)
y_predict_train = lr_1_l2.predict(scaled_X_train)
print(f"MSE on Test: {mean_squared_error(y_test, y_predict)**0.5}")
print(f"MSE on Train: {mean_squared_error(y_train, y_predict_train)**0.5}")
print('Коэффициент детерминации на тестовой выборке: ', r2_score(y_test, y_predict))
print('Коэффициент детерминации на тренировочной выборке', r2_score(y_train, y_predict_tr
```

```
MSE on Test: 2518.5787904710046
MSE on Train: 1393.8114918752556
Коэффициент детерминации на тестовой выборке:  0.9330823065868887
Коэффициент детерминации на тренировочной выборке 0.959678602411492
```

В [75]:
```
lr_1_l2 = LinearRegression_1(MSEL2Loss(0.5))
lr_1_l2.fit(scaled_X_train, y_train)
y_predict = lr_1_l2.predict(scaled_X_test)
y_predict_train = lr_1_l2.predict(scaled_X_train)
print(f"MSE on Test: {mean_squared_error(y_test, y_predict)**0.5}")
print(f"MSE on Train: {mean_squared_error(y_train, y_predict_train)**0.5}")
print('Коэффициент детерминации на тестовой выборке: ', r2_score(y_test, y_predict))
print('Коэффициент детерминации на тренировочной выборке', r2_score(y_train, y_predict_tr
```

```
MSE on Test: 3197.41511786243
MSE on Train: 1898.5504183773194
Коэффициент детерминации на тестовой выборке:  0.8921480684437594
Коэффициент детерминации на тренировочной выборке 0.9251879139460245
```

В [76]:
```
lr_1_l2 = LinearRegression_1(MSEL2Loss(0.9))
lr_1_l2.fit(scaled_X_train, y_train)
y_predict = lr_1_l2.predict(scaled_X_test)
y_predict_train = lr_1_l2.predict(scaled_X_train)
print(f"MSE on Test: {mean_squared_error(y_test, y_predict)**0.5}")
print(f"MSE on Train: {mean_squared_error(y_train, y_predict_train)**0.5}")
print('Коэффициент детерминации на тестовой выборке: ', r2_score(y_test, y_predict))
print('Коэффициент детерминации на тренировочной выборке', r2_score(y_train, y_predict_tr
```

```
MSE on Test: 3562.7185617962423
MSE on Train: 2117.8816804070384
Коэффициент детерминации на тестовой выборке:  0.8660961967849566
Коэффициент детерминации на тренировочной выборке 0.9069040297555025
```

В [77]:
```
lr_1_l2 = LinearRegression_1(MSEL2Loss(0.000000001))
lr_1_l2.fit(scaled_X_train, y_train)
y_predict = lr_1_l2.predict(scaled_X_test)
y_predict_train = lr_1_l2.predict(scaled_X_train)
print(f"MSE on Test: {mean_squared_error(y_test, y_predict)**0.5}")
print(f"MSE on Train: {mean_squared_error(y_train, y_predict_train)**0.5}")
print('Коэффициент детерминации на тестовой выборке: ', r2_score(y_test, y_predict))
print('Коэффициент детерминации на тренировочной выборке', r2_score(y_train, y_predict_tr
```

```
MSE on Test: 2518.7941514179693
MSE on Train: 1393.8159172538788
Коэффициент детерминации на тестовой выборке:  0.9330708619785003
Коэффициент детерминации на тренировочной выборке 0.9596783463686409
```

Улучшить результат при изменении коэффициента не получилось. В каких-то случаях может падать значение коэффициента качестве всех значений падало