

```
{
  "authors": [
    {
      "name": "Алексеев Андрей Сергеевич"
    }
  ],
  "group": "ИУ5-62Б",
  "kernel_spec": {
    "name": "python3",
    "display_name": "Python 3 (ipykernel)",
    "language": "python"
  },
  "language_info": {
    "name": "python",
    "version": "3.9.7",
    "mimetype": "text/x-python",
    "codemirror_mode": {
      "name": "ipython",
      "version": 3
    },
    "pygments_lexer": "ipython3",
    "nbconvert_exporter": "python",
    "file_extension": ".py"
  },
  "title": "Подготовка обучающей и тестовой выборки, кросс-валидация и
подбор гиперпараметров на примере метода ближайших соседей"
}
```

```
from operator import itemgetter
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import numpy as np
import math
from sklearn.model_selection import train_test_split

from enum import Enum
class PredictionType(Enum):
    CLASSIFICATION = 1
    REGRESSION = 2

class SimpleKNN:

    def fit(self, X_train: np.matrix, y_train: np.ndarray):
        """
        Метод обучения, который фактически не учится,
        а только запоминает обучающую выборку.
        Входные параметры:
        X_train - обучающая выборка (матрица объект-признак)
        y_train - обучающая выборка (вектор целевого признака)
        Возвращаемое значение: нет
        """
```

```

        #Сохраняем параметры в переменных класса
        self._X_train = X_train
        self._y_train = y_train

    def eucl_dist(self, p: np.ndarray, q: np.ndarray) -> float:
        """
        Вычисление Евклидова расстояния -
        https://en.wikipedia.org/wiki/Euclidean\_distance
        Входные параметры:
        p, q - вектора в n-мерном пространстве признаков
        """
        return math.sqrt(sum([(pi - qi) ** 2 for pi, qi in zip (p,
q)]))

    def predict_for_single_object(self, K: int, \
        prediction_type: PredictionType, \
        X_o: np.ndarray, \
        verbose = True) -> np.ndarray:
        """
        Метод предсказания для одного объекта.
        Входные параметры:
        K - гиперпараметр, количество соседей
        prediction_type - классификация или регрессия
        X_o - строка матрицы объект-признак, соответствующая объекту
        verbose - флаг детального вывода
        Возвращаемое значение: предсказанное значение целевого
признака
        """
        # список соседей
        neighbors_list = []
        # *** Находим ближайшие точки ***
        # Перебираем все точки обучающей выборки
        for i in range(self._X_train.shape[0]):
            # получаем текущую точку
            data_train_current_x = [x for x in
self._X_train[['high','low']].iloc[i]]
            # и значение ее y
            data_train_current_y = self._y_train[i]
            # вычисляем расстояние
            dist = self.eucl_dist(X_o, data_train_current_x)
            # сохраняем в список соседей
            temp_res = (data_train_current_y, dist,
data_train_current_x)
            neighbors_list.append(temp_res)
        # *** сортируем список соседей по возрастанию расстояния ***
        # в кортеже элементы следуют в порядке (0,1,2), сортируем по
первому элементу
        neighbors_list_sorted = sorted(neighbors_list,
key=itemgetter(1))

```

```

    if verbose:
        print()
        print('*****')
        print('Проверяемая точка: ', X_o)
        print('*****')
        print('Вывод отсортированного списка соседей:')
        dist_list = []
        for cur_y, cur_dist, temp_x_1_2 in neighbors_list_sorted:
            temp_x1, temp_x2 = temp_x_1_2
            print('X1={0}, X2={1}, y={2},
расстояние={3:.2f}'.format(temp_x1, temp_x2, cur_y, cur_dist))
            dist_list.append(cur_dist)
        print()
        print('Вывод расстояния для отсортированного списка
соседей:')
        plt.plot(dist_list)
        plt.show()
        # Оставим только K ближайших соседей
        K_neighbors_list_sorted = neighbors_list_sorted[:K]
        if verbose:
            print('Вывод K ближайших соседей:')
            x1_list = []
            x2_list = []
            for cur_y, cur_dist, temp_x_1_2 in
K_neighbors_list_sorted:
                temp_x1, temp_x2 = temp_x_1_2
                x1_list.append(temp_x1)
                x2_list.append(temp_x2)
                print('X1={0}, X2={1}, y={2},
расстояние={3:.2f}'.format(temp_x1, temp_x2, cur_y, cur_dist))
            print()
            print('Визуализация K ближайших соседей:')
            plt.plot(self.X_train['high'], self.X_train['low'],
'b.', \
                    x1_list, x2_list, 'g*', \
                    [X_o[0], [X_o[1]], 'ro')
            plt.show()
        # Результат - классификация или регрессия
        if prediction_type == PredictionType.REGRESSION:
            # используем numpy для вычисления среднего значения
            arr = np.array([x for x,_,_ in K_neighbors_list_sorted])
            # возвращаем среднее значение
            return np.mean(arr)
        elif prediction_type == PredictionType.CLASSIFICATION:
            k_y_list = [y for y,_,_ in K_neighbors_list_sorted]
            # группируем с количеством метки классов,
            # соответствующие K ближайшим соседям
            k_y_list_grouped_temp = np.unique(k_y_list,
return_counts=True)
            k_y_list_grouped = [[key, cnt] for key, cnt in

```


1	A	1999-11-19	42.94	43.00	39.81	40.38	10897100.0
0.0							
2	A	1999-11-22	41.31	44.00	40.06	44.00	4705200.0
0.0							
3	A	1999-11-23	42.50	43.63	40.25	40.25	4274400.0
0.0							
4	A	1999-11-24	40.13	41.94	40.00	41.06	3464400.0
0.0							
5	A	1999-11-26	40.88	41.50	40.75	41.19	1237100.0
0.0							
6	A	1999-11-29	41.00	42.44	40.56	42.13	2914700.0
0.0							
7	A	1999-11-30	42.00	42.94	40.94	42.19	3083000.0
0.0							
8	A	1999-12-01	42.19	43.44	41.88	42.94	2115400.0
0.0							
9	A	1999-12-02	43.75	45.00	43.19	44.13	2195900.0
0.0							
10	A	1999-12-03	44.94	45.69	44.31	44.50	2175700.0
0.0							
11	A	1999-12-06	45.25	46.44	45.19	45.75	1610000.0
0.0							
12	A	1999-12-07	45.75	46.00	44.31	45.25	1585100.0
0.0							
13	A	1999-12-08	45.25	45.63	44.81	45.19	1350400.0
0.0							
14	A	1999-12-09	45.25	45.94	45.25	45.81	1451400.0
0.0							
15	A	1999-12-10	45.69	45.94	44.75	44.75	1190800.0
0.0							
16	A	1999-12-13	45.50	46.25	44.38	45.50	2875900.0
0.0							
17	A	1999-12-14	45.38	45.38	42.06	43.00	1665900.0
0.0							
18	A	1999-12-15	42.00	42.31	41.00	41.69	2087100.0
0.0							
19	A	1999-12-16	42.00	48.00	42.00	47.25	1848300.0
0.0							

	split_ratio	adj_open	adj_high	adj_low	adj_close
adj_volume					
0	1.0	31.041951	34.112034	27.289627	30.018590
44739900.0					
1	1.0	29.295415	29.336350	27.160002	27.548879
10897100.0					
2	1.0	28.183363	30.018590	27.330562	30.018590
4705200.0					
3	1.0	28.995229	29.766161	27.460188	27.460188
4274400.0					
4	1.0	27.378319	28.613174	27.289627	28.012803

3464400.0					
5	1.0	27.889999	28.312988	27.801308	28.101494
1237100.0					
6	1.0	27.971868	28.954295	27.671682	28.742800
2914700.0					
7	1.0	28.654109	29.295415	27.930934	28.783735
3083000.0					
8	1.0	28.783735	29.636535	28.572240	29.295415
2115400.0					
9	1.0	29.848030	30.700831	29.465975	30.107281
2195900.0					
10	1.0	30.659896	31.171577	30.230085	30.359711
2175700.0					
11	1.0	30.871391	31.683257	30.830457	31.212511
1610000.0					
12	1.0	31.212511	31.383072	30.230085	30.871391
1585100.0					
13	1.0	30.871391	31.130643	30.571205	30.830457
1350400.0					
14	1.0	30.871391	31.342137	30.871391	31.253446
1451400.0					
15	1.0	31.171577	31.342137	30.530271	30.530271
1190800.0					
16	1.0	31.041951	31.553632	30.277842	31.041951
2875900.0					
17	1.0	30.960082	30.960082	28.695043	29.336350
1665900.0					
18	1.0	28.654109	28.865603	27.971868	28.442614
2087100.0					
19	1.0	28.654109	32.747553	28.654109	32.235872
1848300.0					

```
def regr_to_class(high: float) -> str:
    if high<43:
        result = 'A'
    elif high<46:
        result = 'B'
    elif high<50:
        result = 'C'
    else:
        result = 'D'
    return result
```

формирование второго целевого признака для классификации

```
data_train['y_clas'] = \
data_train.apply(lambda row: regr_to_class(row['high']),axis=1)
```

```
data_train
```

```
    ticker      date  open  high  low  close      volume  ex-
dividend \
```

0	A	1999-11-18	45.50	50.00	40.00	44.00	44739900.0
0.0							
1	A	1999-11-19	42.94	43.00	39.81	40.38	10897100.0
0.0							
2	A	1999-11-22	41.31	44.00	40.06	44.00	4705200.0
0.0							
3	A	1999-11-23	42.50	43.63	40.25	40.25	4274400.0
0.0							
4	A	1999-11-24	40.13	41.94	40.00	41.06	3464400.0
0.0							
5	A	1999-11-26	40.88	41.50	40.75	41.19	1237100.0
0.0							
6	A	1999-11-29	41.00	42.44	40.56	42.13	2914700.0
0.0							
7	A	1999-11-30	42.00	42.94	40.94	42.19	3083000.0
0.0							
8	A	1999-12-01	42.19	43.44	41.88	42.94	2115400.0
0.0							
9	A	1999-12-02	43.75	45.00	43.19	44.13	2195900.0
0.0							
10	A	1999-12-03	44.94	45.69	44.31	44.50	2175700.0
0.0							
11	A	1999-12-06	45.25	46.44	45.19	45.75	1610000.0
0.0							
12	A	1999-12-07	45.75	46.00	44.31	45.25	1585100.0
0.0							
13	A	1999-12-08	45.25	45.63	44.81	45.19	1350400.0
0.0							
14	A	1999-12-09	45.25	45.94	45.25	45.81	1451400.0
0.0							
15	A	1999-12-10	45.69	45.94	44.75	44.75	1190800.0
0.0							
16	A	1999-12-13	45.50	46.25	44.38	45.50	2875900.0
0.0							
17	A	1999-12-14	45.38	45.38	42.06	43.00	1665900.0
0.0							
18	A	1999-12-15	42.00	42.31	41.00	41.69	2087100.0
0.0							
19	A	1999-12-16	42.00	48.00	42.00	47.25	1848300.0
0.0							

	split_ratio	adj_open	adj_high	adj_low	adj_close
adj_volume	y_clas				
0	1.0	31.041951	34.112034	27.289627	30.018590
44739900.0	D				
1	1.0	29.295415	29.336350	27.160002	27.548879
10897100.0	B				
2	1.0	28.183363	30.018590	27.330562	30.018590
4705200.0	B				
3	1.0	28.995229	29.766161	27.460188	27.460188

4274400.0	B				
4	1.0	27.378319	28.613174	27.289627	28.012803
3464400.0	A				
5	1.0	27.889999	28.312988	27.801308	28.101494
1237100.0	A				
6	1.0	27.971868	28.954295	27.671682	28.742800
2914700.0	A				
7	1.0	28.654109	29.295415	27.930934	28.783735
3083000.0	A				
8	1.0	28.783735	29.636535	28.572240	29.295415
2115400.0	B				
9	1.0	29.848030	30.700831	29.465975	30.107281
2195900.0	B				
10	1.0	30.659896	31.171577	30.230085	30.359711
2175700.0	B				
11	1.0	30.871391	31.683257	30.830457	31.212511
1610000.0	C				
12	1.0	31.212511	31.383072	30.230085	30.871391
1585100.0	C				
13	1.0	30.871391	31.130643	30.571205	30.830457
1350400.0	B				
14	1.0	30.871391	31.342137	30.871391	31.253446
1451400.0	B				
15	1.0	31.171577	31.342137	30.530271	30.530271
1190800.0	B				
16	1.0	31.041951	31.553632	30.277842	31.041951
2875900.0	C				
17	1.0	30.960082	30.960082	28.695043	29.336350
1665900.0	B				
18	1.0	28.654109	28.865603	27.971868	28.442614
2087100.0	A				
19	1.0	28.654109	32.747553	28.654109	32.235872
1848300.0	C				

чтение тестовой выборки

```
data test = pd.read_csv('WIKI PRICES test.csv', sep=",")
```

data_test

ticker	date	open	high	low	close	volume	ex-
dividend \							
0	A	1999-12-17	46.38	47.12	45.44	45.94	2652400.0
0.0							
1	A	1999-12-20	46.25	46.94	46.13	46.88	856100.0
0.0							
2	A	1999-12-21	46.69	46.69	46.00	46.63	1616200.0
0.0							
3	A	1999-12-22	46.63	47.56	46.31	47.56	1363200.0
0.0							
4	A	1999-12-23	47.50	50.00	47.44	49.75	1544700.0
0.0							

	split_ratio	adj_open	adj_high	adj_low	adj_close	adj_volume
0	1.0	31.642323	32.147181	31.001017	31.342137	2652400.0
1	1.0	31.553632	32.024378	31.471763	31.983443	856100.0
2	1.0	31.853818	31.853818	31.383072	31.812883	1616200.0
3	1.0	31.812883	32.447367	31.594566	32.447367	1363200.0
4	1.0	32.406433	34.112034	32.365498	33.941474	1544700.0

визуализация выборки

```
fig, ax = plt.subplots(figsize=(8,8))
```

```
ax.plot(data_train['high'], data_train['low'], 'b.', \
        data_test['high'], data_test['low'], 'ro')
```

деления на осях и сетка

```
ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
```

```
ax.yaxis.set_major_locator(ticker.MultipleLocator(1))
```

```
ax.grid(which='major', color = 'k')
```

подписи к осям

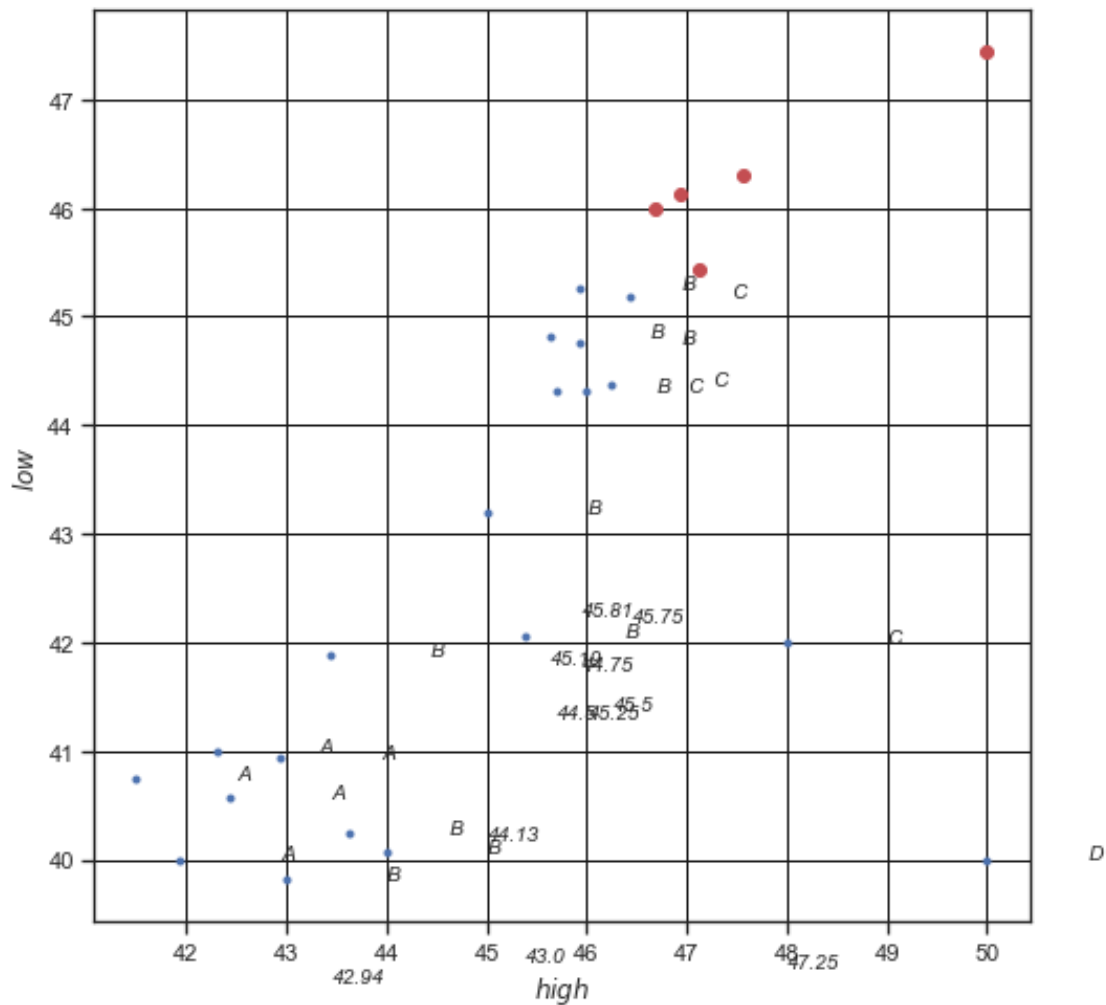
```
plt.xlabel('$high$')
```

```
plt.ylabel('$low$')
```

подписи

```
for coords in data_train[['high','low', 'close', 'y_clas']].values:
    high, low, y_cl, cl = coords[0], coords[1], coords[2], coords[3]
    #label = '{} ({}).format(y_cl, cl)
    #ax.text(high + 1, low, label, style='italic', fontsize=7)
    ax.text(high + 1, low, cl, style='italic', fontsize=10)
    ax.text(high, low-3, y_cl, style='italic', fontsize=10)
```

```
plt.show()
```



```
# создаем объект класса
simple_knn_regr = SimpleKNN()

# вызываем метод обучения
simple_knn_regr.fit(data_train[['high', 'low']], data_train['close'])

# параметры метода
# матрица объект-признак
simple_knn_regr.X_train
```

	high	low
0	50.00	40.00
1	43.00	39.81
2	44.00	40.06

3	43.63	40.25
4	41.94	40.00
5	41.50	40.75
6	42.44	40.56
7	42.94	40.94
8	43.44	41.88
9	45.00	43.19
10	45.69	44.31
11	46.44	45.19
12	46.00	44.31
13	45.63	44.81
14	45.94	45.25
15	45.94	44.75
16	46.25	44.38
17	45.38	42.06
18	42.31	41.00
19	48.00	42.00

вектор целевого признака
simple_knn_regr._y_train

0	44.00
1	40.38
2	44.00
3	40.25
4	41.06
5	41.19
6	42.13
7	42.19
8	42.94
9	44.13
10	44.50
11	45.75
12	45.25
13	45.19
14	45.81
15	44.75
16	45.50
17	43.00
18	41.69
19	47.25

Name: close, dtype: float64

simple_knn_clas = SimpleKNN()
simple_knn_clas.fit(data_train[['high', 'low']], data_train['y_clas'])

simple_knn_clas._X_train

	high	low
0	50.00	40.00
1	43.00	39.81

```
2    44.00    40.06
3    43.63    40.25
4    41.94    40.00
5    41.50    40.75
6    42.44    40.56
7    42.94    40.94
8    43.44    41.88
9    45.00    43.19
10   45.69    44.31
11   46.44    45.19
12   46.00    44.31
13   45.63    44.81
14   45.94    45.25
15   45.94    44.75
16   46.25    44.38
17   45.38    42.06
18   42.31    41.00
19   48.00    42.00
```

```
simple_knn_clas._y_train
```

```
0    D
1    B
2    B
3    B
4    A
5    A
6    A
7    A
8    B
9    B
10   B
11   C
12   C
13   B
14   B
15   B
16   C
17   B
18   A
19   C
```

```
Name: y_clas, dtype: object
```

```
# первая строка тестовой выборки
```

```
data_test_0 = [x for x in data_test[['high','low']].iloc[0]]
data_test_0
```

```
[47.12, 45.44]
```

```
# построим предсказание для одного объекта (регрессия)
```

```
simple_knn_regr_0 = simple_knn_regr.predict_for_single_object(K=5, \
```

```
prediction_type=PredictionType.REGRESSION, \
X_o=data_test_0)

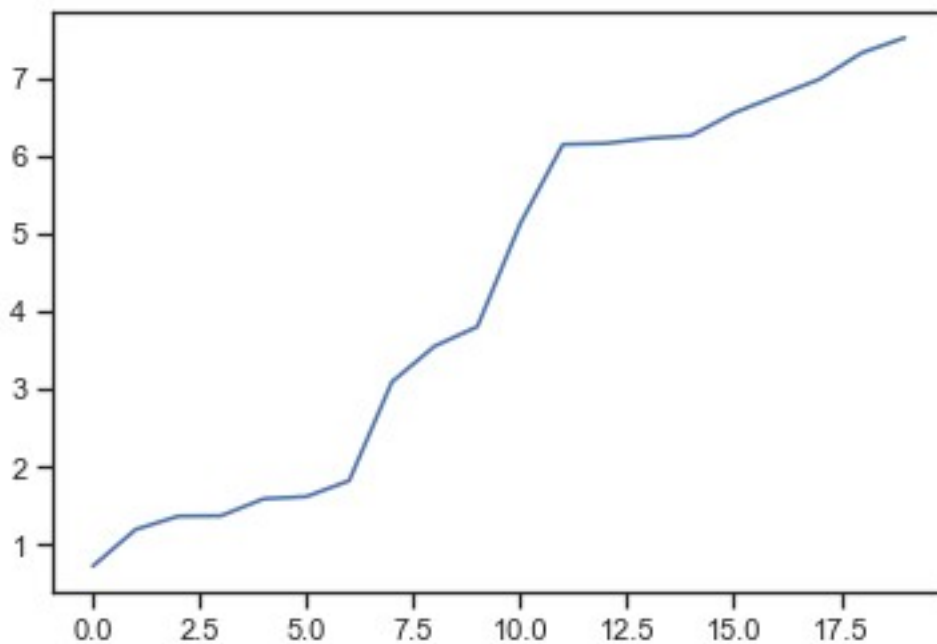
simple_knn_regr_0
```

Проверяемая точка: [47.12, 45.44]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=0.72
X1=45.94, X2=45.25, y=45.81, расстояние=1.20
X1=45.94, X2=44.75, y=44.75, расстояние=1.37
X1=46.25, X2=44.38, y=45.5, расстояние=1.37
X1=46.0, X2=44.31, y=45.25, расстояние=1.59
X1=45.63, X2=44.81, y=45.19, расстояние=1.62
X1=45.69, X2=44.31, y=44.5, расстояние=1.82
X1=45.0, X2=43.19, y=44.13, расстояние=3.09
X1=48.0, X2=42.0, y=47.25, расстояние=3.55
X1=45.38, X2=42.06, y=43.0, расстояние=3.80
X1=43.44, X2=41.88, y=42.94, расстояние=5.12
X1=42.94, X2=40.94, y=42.19, расстояние=6.14
X1=50.0, X2=40.0, y=44.0, расстояние=6.16
X1=44.0, X2=40.06, y=44.0, расстояние=6.22
X1=43.63, X2=40.25, y=40.25, расстояние=6.25
X1=42.31, X2=41.0, y=41.69, расстояние=6.55
X1=42.44, X2=40.56, y=42.13, расстояние=6.76
X1=43.0, X2=39.81, y=40.38, расстояние=6.98
X1=41.5, X2=40.75, y=41.19, расстояние=7.32
X1=41.94, X2=40.0, y=41.06, расстояние=7.51

Вывод расстояния для отсортированного списка соседей:



Вывод К ближайших соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=0.72

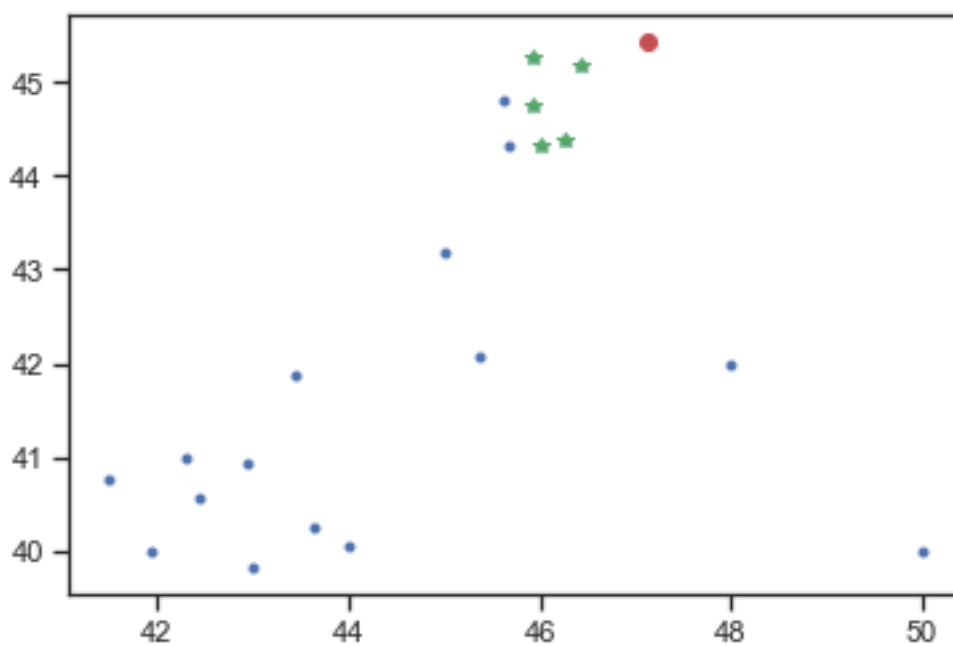
X1=45.94, X2=45.25, y=45.81, расстояние=1.20

X1=45.94, X2=44.75, y=44.75, расстояние=1.37

X1=46.25, X2=44.38, y=45.5, расстояние=1.37

X1=46.0, X2=44.31, y=45.25, расстояние=1.59

Визуализация К ближайших соседей:



45.412

```
# Среднее значение y для соседей
np.mean([45.75, 45.81, 44.75, 45.5, 45.25])

45.412

# построим предсказание для одного объекта (классификация)
simple_knn_clas_0 = simple_knn_clas.predict_for_single_object(K=5, \

prediction_type=PredictionType.CLASSIFICATION, \
                    X_o=data_test_0)

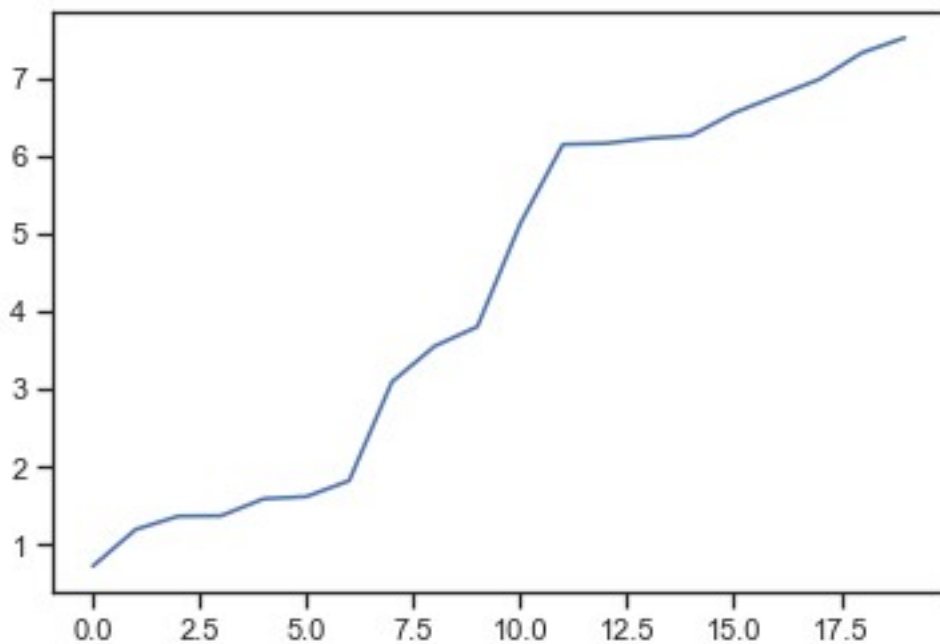
simple_knn_clas_0
```

Проверяемая точка: [47.12, 45.44]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=C, расстояние=0.72
X1=45.94, X2=45.25, y=B, расстояние=1.20
X1=45.94, X2=44.75, y=B, расстояние=1.37
X1=46.25, X2=44.38, y=C, расстояние=1.37
X1=46.0, X2=44.31, y=C, расстояние=1.59
X1=45.63, X2=44.81, y=B, расстояние=1.62
X1=45.69, X2=44.31, y=B, расстояние=1.82
X1=45.0, X2=43.19, y=B, расстояние=3.09
X1=48.0, X2=42.0, y=C, расстояние=3.55
X1=45.38, X2=42.06, y=B, расстояние=3.80
X1=43.44, X2=41.88, y=B, расстояние=5.12
X1=42.94, X2=40.94, y=A, расстояние=6.14
X1=50.0, X2=40.0, y=D, расстояние=6.16
X1=44.0, X2=40.06, y=B, расстояние=6.22
X1=43.63, X2=40.25, y=B, расстояние=6.25
X1=42.31, X2=41.0, y=A, расстояние=6.55
X1=42.44, X2=40.56, y=A, расстояние=6.76
X1=43.0, X2=39.81, y=B, расстояние=6.98
X1=41.5, X2=40.75, y=A, расстояние=7.32
X1=41.94, X2=40.0, y=A, расстояние=7.51

Вывод расстояния для отсортированного списка соседей:



Вывод K ближайших соседей:

X1=46.44, X2=45.19, y=C, расстояние=0.72

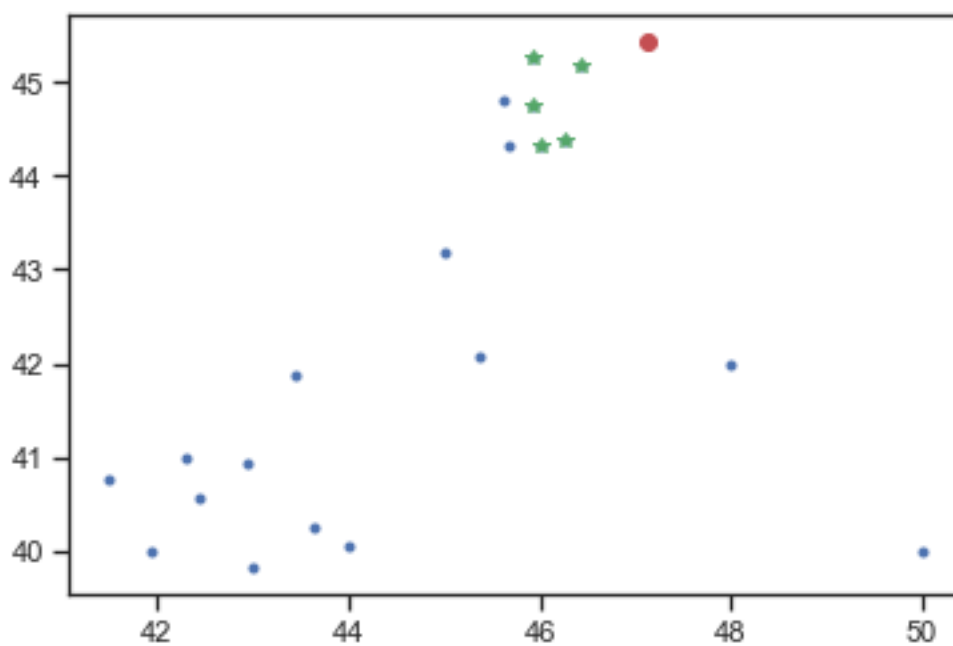
X1=45.94, X2=45.25, y=B, расстояние=1.20

X1=45.94, X2=44.75, y=B, расстояние=1.37

X1=46.25, X2=44.38, y=C, расстояние=1.37

X1=46.0, X2=44.31, y=C, расстояние=1.59

Визуализация K ближайших соседей:



Классы, соответствующие K ближайшим соседям:

класс=C, количество элементов=3

класс=B, количество элементов=2

'C'

Для всех объектов тестовой выборки (регрессия)

```
simple_knn_regr_prediction = simple_knn_regr.predict(K=5, \
                                                    prediction_type=PredictionType.REGRESSION, \
                                                    X_test=data_test[['high', 'low']])
```

simple_knn_regr_prediction

Проверяемая точка: [47.12, 45.44]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=0.72

X1=45.94, X2=45.25, y=45.81, расстояние=1.20

X1=45.94, X2=44.75, y=44.75, расстояние=1.37

X1=46.25, X2=44.38, y=45.5, расстояние=1.37

X1=46.0, X2=44.31, y=45.25, расстояние=1.59

X1=45.63, X2=44.81, y=45.19, расстояние=1.62

X1=45.69, X2=44.31, y=44.5, расстояние=1.82

X1=45.0, X2=43.19, y=44.13, расстояние=3.09

X1=48.0, X2=42.0, y=47.25, расстояние=3.55

X1=45.38, X2=42.06, y=43.0, расстояние=3.80

X1=43.44, X2=41.88, y=42.94, расстояние=5.12

X1=42.94, X2=40.94, y=42.19, расстояние=6.14

X1=50.0, X2=40.0, y=44.0, расстояние=6.16

X1=44.0, X2=40.06, y=44.0, расстояние=6.22

X1=43.63, X2=40.25, y=40.25, расстояние=6.25

X1=42.31, X2=41.0, y=41.69, расстояние=6.55

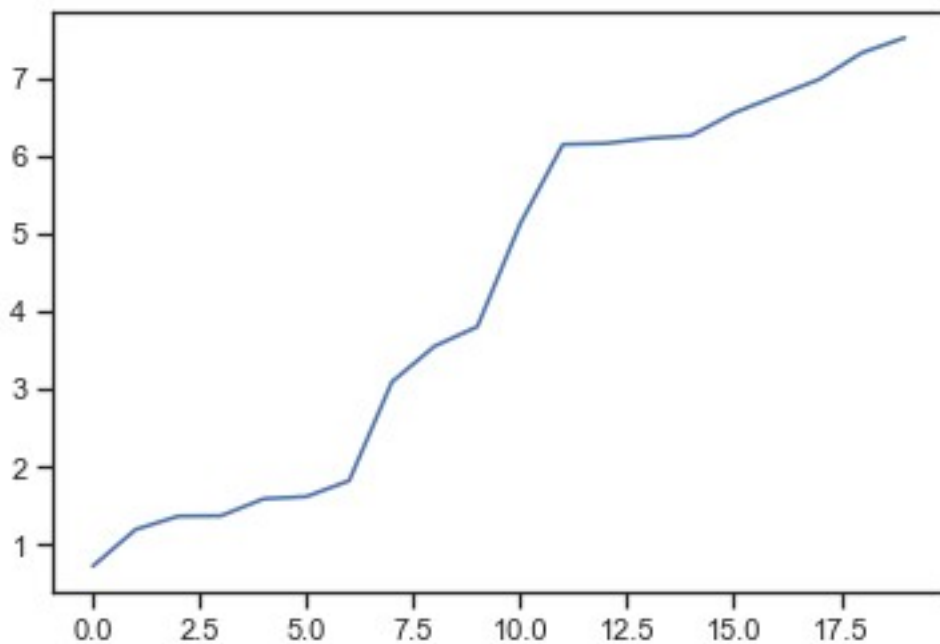
X1=42.44, X2=40.56, y=42.13, расстояние=6.76

X1=43.0, X2=39.81, y=40.38, расстояние=6.98

X1=41.5, X2=40.75, y=41.19, расстояние=7.32

X1=41.94, X2=40.0, y=41.06, расстояние=7.51

Вывод расстояния для отсортированного списка соседей:



Вывод К ближайших соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=0.72

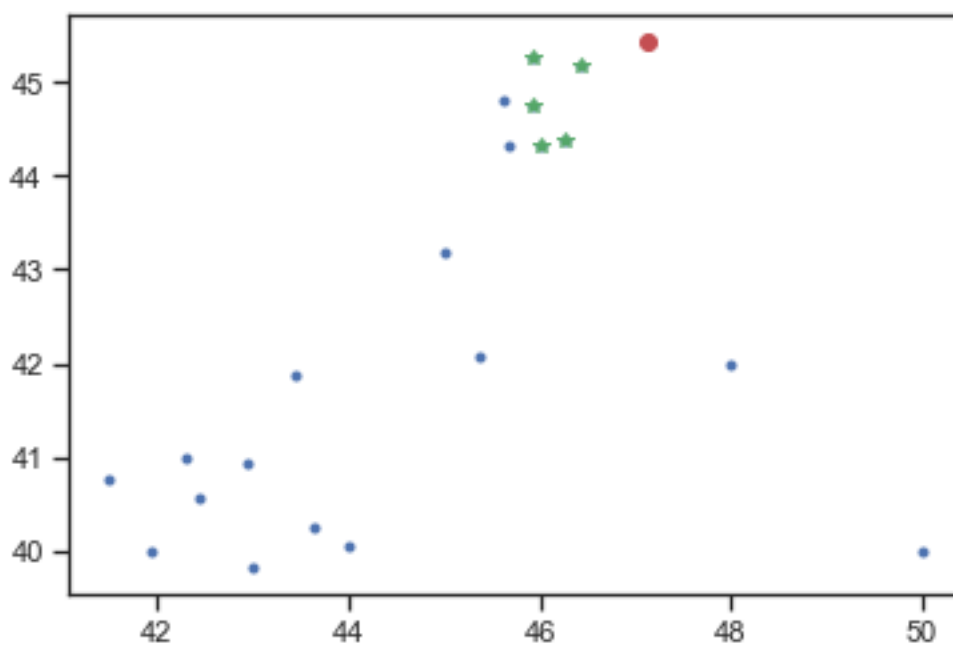
X1=45.94, X2=45.25, y=45.81, расстояние=1.20

X1=45.94, X2=44.75, y=44.75, расстояние=1.37

X1=46.25, X2=44.38, y=45.5, расстояние=1.37

X1=46.0, X2=44.31, y=45.25, расстояние=1.59

Визуализация К ближайших соседей:

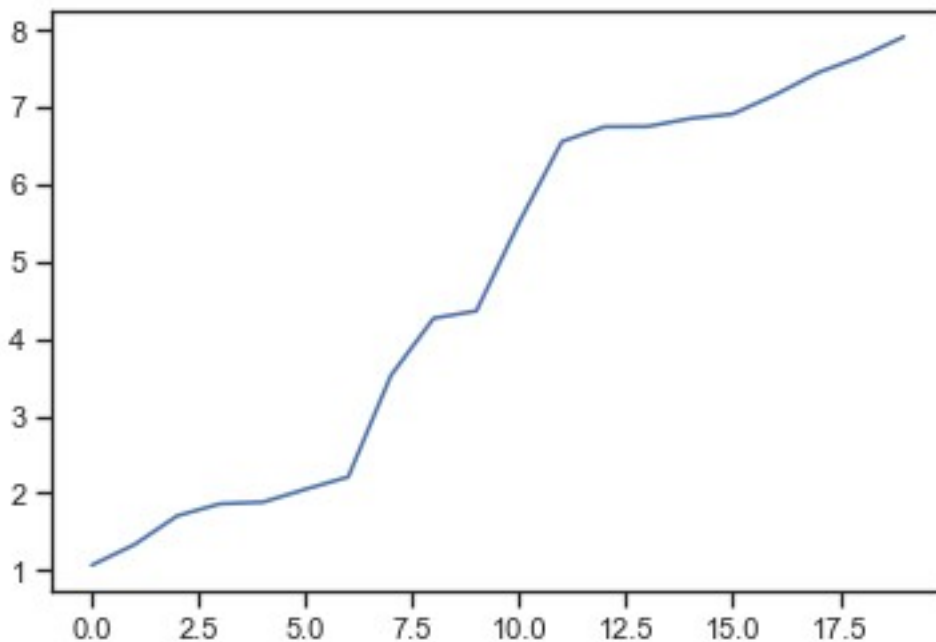


Проверяемая точка: [46.94, 46.13]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=1.06
X1=45.94, X2=45.25, y=45.81, расстояние=1.33
X1=45.94, X2=44.75, y=44.75, расстояние=1.70
X1=45.63, X2=44.81, y=45.19, расстояние=1.86
X1=46.25, X2=44.38, y=45.5, расстояние=1.88
X1=46.0, X2=44.31, y=45.25, расстояние=2.05
X1=45.69, X2=44.31, y=44.5, расстояние=2.21
X1=45.0, X2=43.19, y=44.13, расстояние=3.52
X1=48.0, X2=42.0, y=47.25, расстояние=4.26
X1=45.38, X2=42.06, y=43.0, расстояние=4.36
X1=43.44, X2=41.88, y=42.94, расстояние=5.51
X1=42.94, X2=40.94, y=42.19, расстояние=6.55
X1=44.0, X2=40.06, y=44.0, расстояние=6.74
X1=43.63, X2=40.25, y=40.25, расстояние=6.75
X1=50.0, X2=40.0, y=44.0, расстояние=6.85
X1=42.31, X2=41.0, y=41.69, расстояние=6.91
X1=42.44, X2=40.56, y=42.13, расстояние=7.16
X1=43.0, X2=39.81, y=40.38, расстояние=7.45
X1=41.5, X2=40.75, y=41.19, расстояние=7.65
X1=41.94, X2=40.0, y=41.06, расстояние=7.91

Вывод расстояния для отсортированного списка соседей:

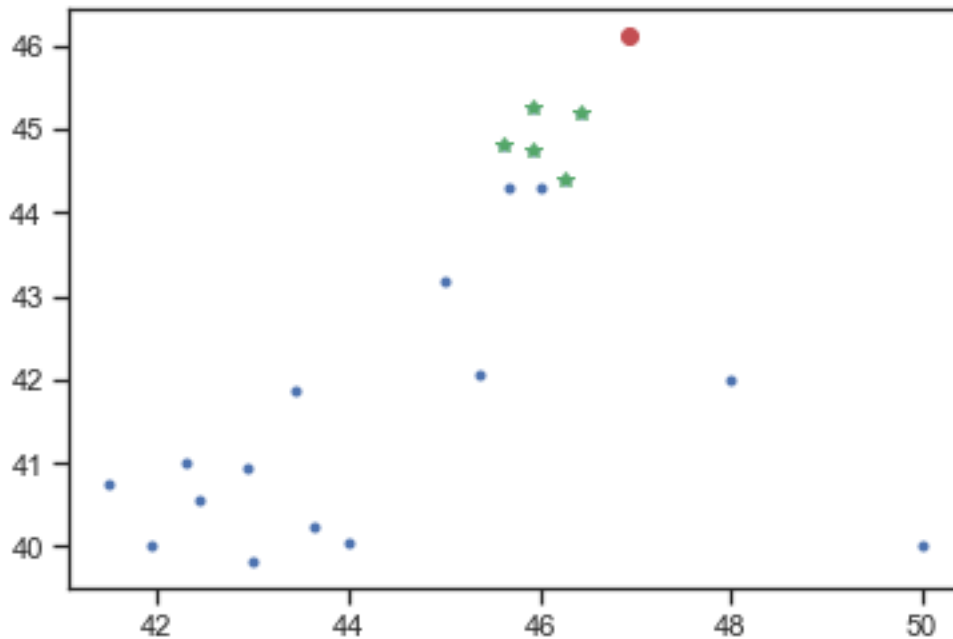


Вывод K ближайших соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=1.06

X1=45.94, X2=45.25, y=45.81, расстояние=1.33
X1=45.94, X2=44.75, y=44.75, расстояние=1.70
X1=45.63, X2=44.81, y=45.19, расстояние=1.86
X1=46.25, X2=44.38, y=45.5, расстояние=1.88

Визуализация К ближайших соседей:



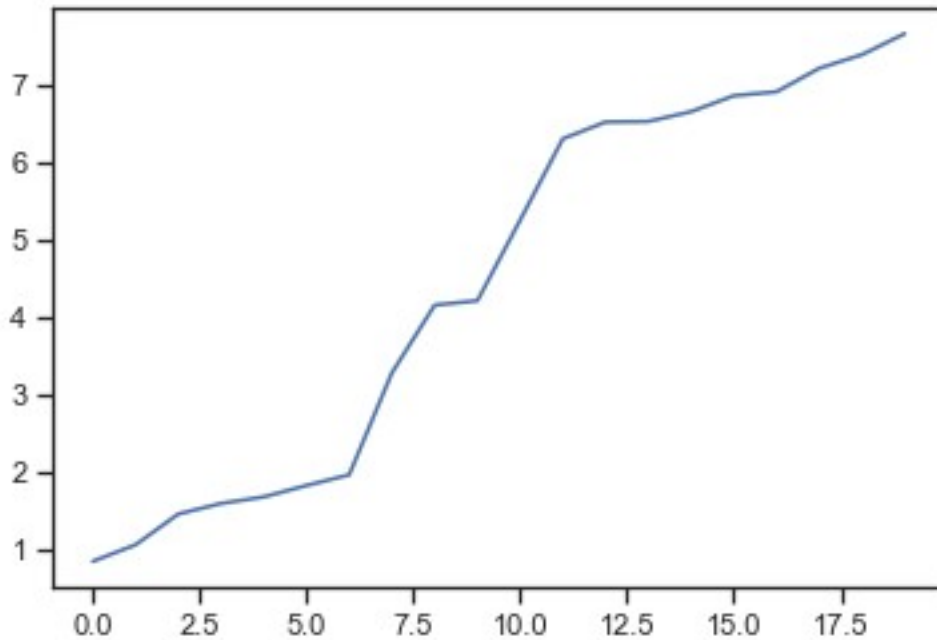
Проверяемая точка: [46.69, 46.0]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=0.85
X1=45.94, X2=45.25, y=45.81, расстояние=1.06
X1=45.94, X2=44.75, y=44.75, расстояние=1.46
X1=45.63, X2=44.81, y=45.19, расстояние=1.59
X1=46.25, X2=44.38, y=45.5, расстояние=1.68
X1=46.0, X2=44.31, y=45.25, расстояние=1.83
X1=45.69, X2=44.31, y=44.5, расстояние=1.96
X1=45.0, X2=43.19, y=44.13, расстояние=3.28
X1=45.38, X2=42.06, y=43.0, расстояние=4.15
X1=48.0, X2=42.0, y=47.25, расстояние=4.21
X1=43.44, X2=41.88, y=42.94, расстояние=5.25
X1=42.94, X2=40.94, y=42.19, расстояние=6.30
X1=43.63, X2=40.25, y=40.25, расстояние=6.51
X1=44.0, X2=40.06, y=44.0, расстояние=6.52
X1=42.31, X2=41.0, y=41.69, расстояние=6.65
X1=50.0, X2=40.0, y=44.0, расстояние=6.85
X1=42.44, X2=40.56, y=42.13, расстояние=6.90
X1=43.0, X2=39.81, y=40.38, расстояние=7.21

X1=41.5, X2=40.75, y=41.19, расстояние=7.38
X1=41.94, X2=40.0, y=41.06, расстояние=7.65

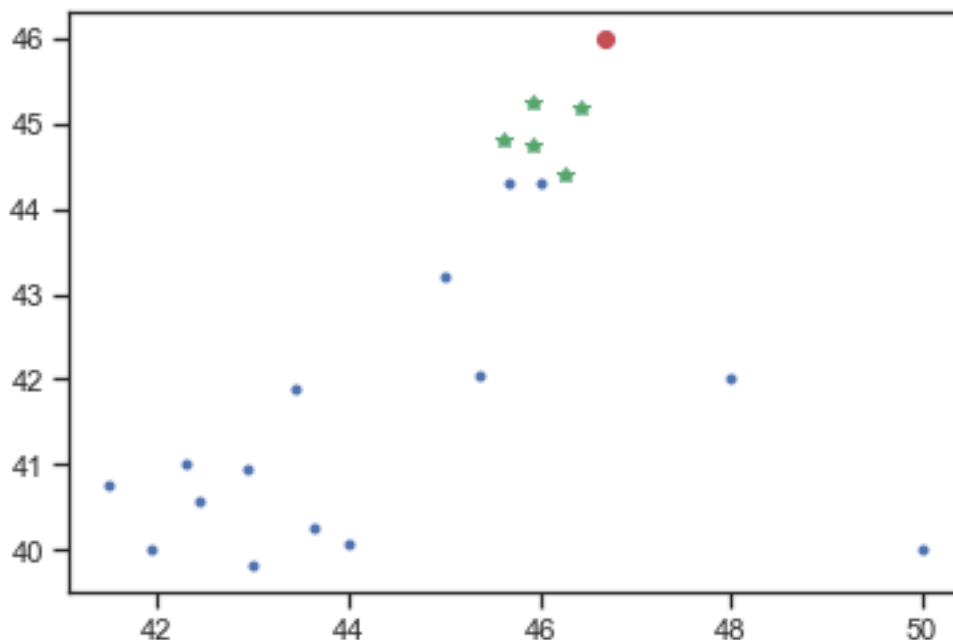
Вывод расстояния для отсортированного списка соседей:



Вывод K ближайших соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=0.85
X1=45.94, X2=45.25, y=45.81, расстояние=1.06
X1=45.94, X2=44.75, y=44.75, расстояние=1.46
X1=45.63, X2=44.81, y=45.19, расстояние=1.59
X1=46.25, X2=44.38, y=45.5, расстояние=1.68

Визуализация K ближайших соседей:

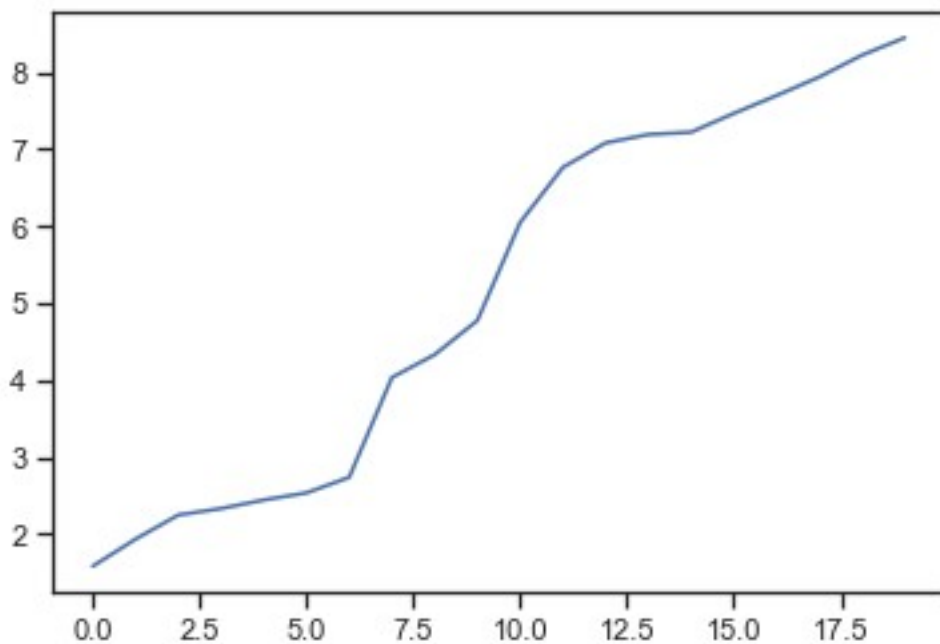


Проверяемая точка: [47.56, 46.31]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=1.58
 X1=45.94, X2=45.25, y=45.81, расстояние=1.94
 X1=45.94, X2=44.75, y=44.75, расстояние=2.25
 X1=46.25, X2=44.38, y=45.5, расстояние=2.33
 X1=45.63, X2=44.81, y=45.19, расстояние=2.44
 X1=46.0, X2=44.31, y=45.25, расстояние=2.54
 X1=45.69, X2=44.31, y=44.5, расстояние=2.74
 X1=45.0, X2=43.19, y=44.13, расстояние=4.04
 X1=48.0, X2=42.0, y=47.25, расстояние=4.33
 X1=45.38, X2=42.06, y=43.0, расстояние=4.78
 X1=43.44, X2=41.88, y=42.94, расстояние=6.05
 X1=50.0, X2=40.0, y=44.0, расстояние=6.77
 X1=42.94, X2=40.94, y=42.19, расстояние=7.08
 X1=44.0, X2=40.06, y=44.0, расстояние=7.19
 X1=43.63, X2=40.25, y=40.25, расстояние=7.22
 X1=42.31, X2=41.0, y=41.69, расстояние=7.47
 X1=42.44, X2=40.56, y=42.13, расстояние=7.70
 X1=43.0, X2=39.81, y=40.38, расстояние=7.94
 X1=41.5, X2=40.75, y=41.19, расстояние=8.22
 X1=41.94, X2=40.0, y=41.06, расстояние=8.45

Вывод расстояния для отсортированного списка соседей:



Вывод K ближайших соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=1.58

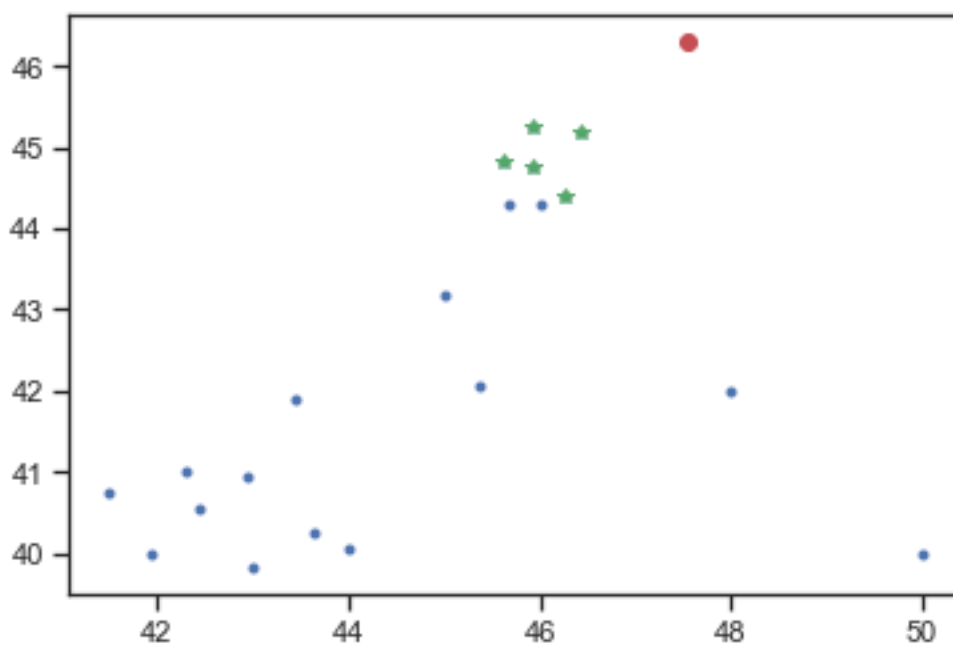
X1=45.94, X2=45.25, y=45.81, расстояние=1.94

X1=45.94, X2=44.75, y=44.75, расстояние=2.25

X1=46.25, X2=44.38, y=45.5, расстояние=2.33

X1=45.63, X2=44.81, y=45.19, расстояние=2.44

Визуализация K ближайших соседей:

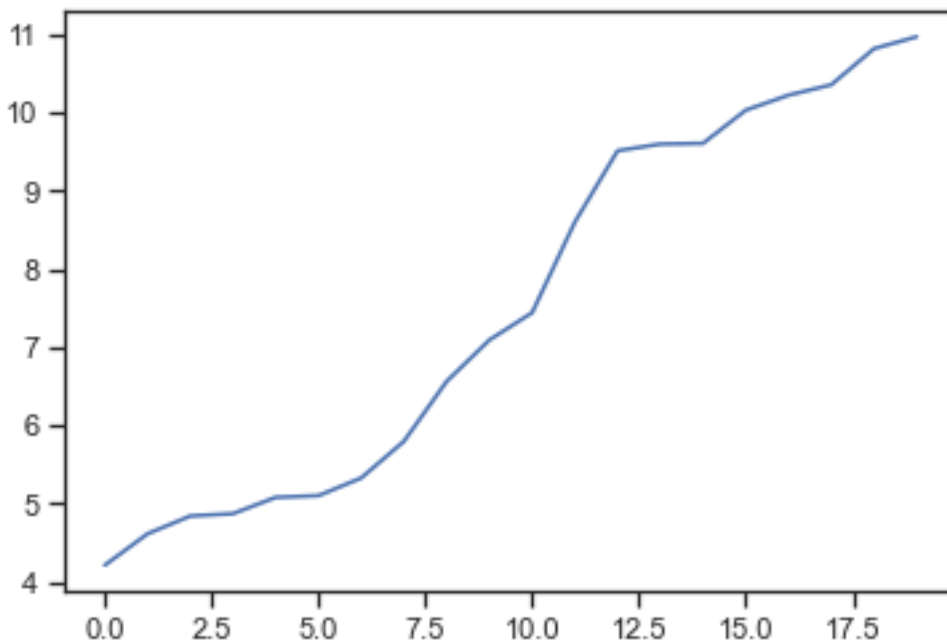


Проверяемая точка: [50.0, 47.44]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=4.21
X1=45.94, X2=45.25, y=45.81, расстояние=4.61
X1=46.25, X2=44.38, y=45.5, расстояние=4.84
X1=45.94, X2=44.75, y=44.75, расстояние=4.87
X1=46.0, X2=44.31, y=45.25, расстояние=5.08
X1=45.63, X2=44.81, y=45.19, расстояние=5.10
X1=45.69, X2=44.31, y=44.5, расстояние=5.33
X1=48.0, X2=42.0, y=47.25, расстояние=5.80
X1=45.0, X2=43.19, y=44.13, расстояние=6.56
X1=45.38, X2=42.06, y=43.0, расстояние=7.09
X1=50.0, X2=40.0, y=44.0, расстояние=7.44
X1=43.44, X2=41.88, y=42.94, расстояние=8.60
X1=44.0, X2=40.06, y=44.0, расстояние=9.51
X1=42.94, X2=40.94, y=42.19, расстояние=9.60
X1=43.63, X2=40.25, y=40.25, расстояние=9.61
X1=42.31, X2=41.0, y=41.69, расстояние=10.03
X1=42.44, X2=40.56, y=42.13, расстояние=10.22
X1=43.0, X2=39.81, y=40.38, расстояние=10.35
X1=41.5, X2=40.75, y=41.19, расстояние=10.82
X1=41.94, X2=40.0, y=41.06, расстояние=10.97

Вывод расстояния для отсортированного списка соседей:

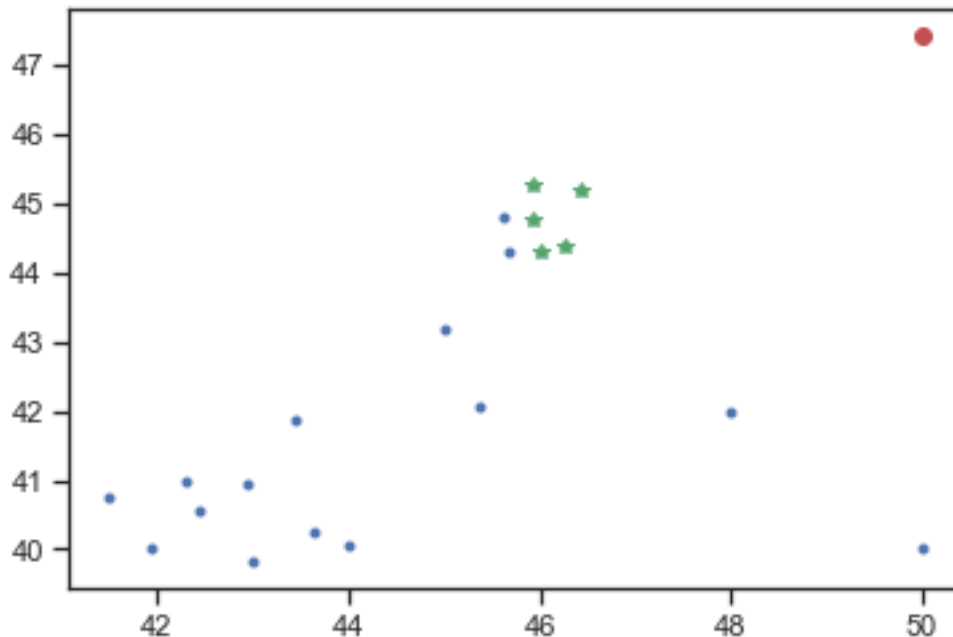


Вывод K ближайших соседей:

X1=46.44, X2=45.19, y=45.75, расстояние=4.21

X1=45.94, X2=45.25, y=45.81, расстояние=4.61
 X1=46.25, X2=44.38, y=45.5, расстояние=4.84
 X1=45.94, X2=44.75, y=44.75, расстояние=4.87
 X1=46.0, X2=44.31, y=45.25, расстояние=5.08

Визуализация К ближайших соседей:



[45.412, 45.4, 45.4, 45.4, 45.412]

```

# Для всех объектов тестовой выборки (классификация)
simple_knn_clas_prediction = simple_knn_clas.predict(K=3, \
prediction_type=PredictionType.CLASSIFICATION, \
X_test=data_test[['high', 'low']])
simple_knn_clas_prediction

```

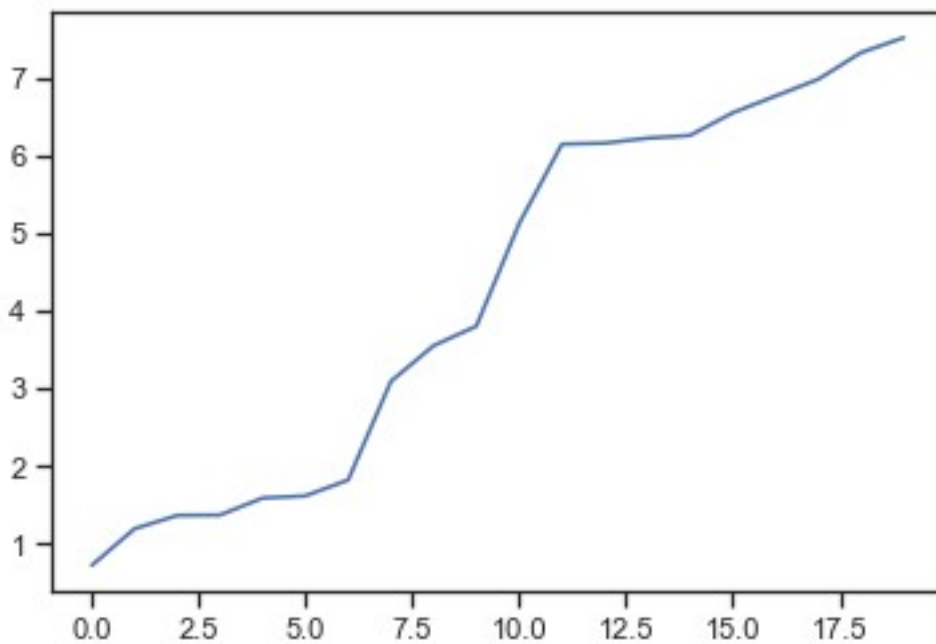
Проверяемая точка: [47.12, 45.44]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=C, расстояние=0.72
 X1=45.94, X2=45.25, y=B, расстояние=1.20
 X1=45.94, X2=44.75, y=B, расстояние=1.37
 X1=46.25, X2=44.38, y=C, расстояние=1.37
 X1=46.0, X2=44.31, y=C, расстояние=1.59
 X1=45.63, X2=44.81, y=B, расстояние=1.62
 X1=45.69, X2=44.31, y=B, расстояние=1.82
 X1=45.0, X2=43.19, y=B, расстояние=3.09
 X1=48.0, X2=42.0, y=C, расстояние=3.55

X1=45.38, X2=42.06, y=B, расстояние=3.80
X1=43.44, X2=41.88, y=B, расстояние=5.12
X1=42.94, X2=40.94, y=A, расстояние=6.14
X1=50.0, X2=40.0, y=D, расстояние=6.16
X1=44.0, X2=40.06, y=B, расстояние=6.22
X1=43.63, X2=40.25, y=B, расстояние=6.25
X1=42.31, X2=41.0, y=A, расстояние=6.55
X1=42.44, X2=40.56, y=A, расстояние=6.76
X1=43.0, X2=39.81, y=B, расстояние=6.98
X1=41.5, X2=40.75, y=A, расстояние=7.32
X1=41.94, X2=40.0, y=A, расстояние=7.51

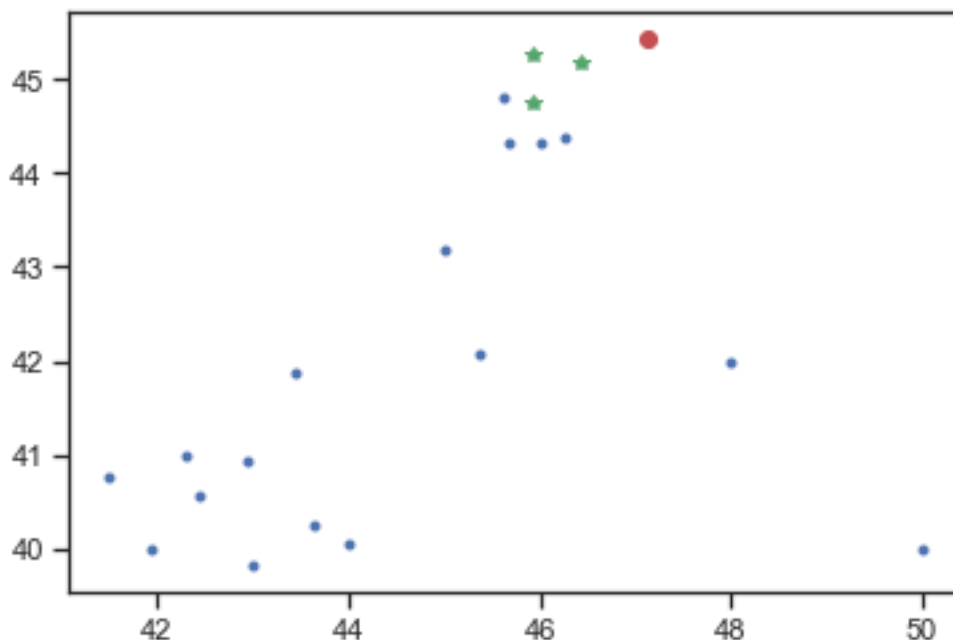
Вывод расстояния для отсортированного списка соседей:



Вывод K ближайших соседей:

X1=46.44, X2=45.19, y=C, расстояние=0.72
X1=45.94, X2=45.25, y=B, расстояние=1.20
X1=45.94, X2=44.75, y=B, расстояние=1.37

Визуализация K ближайших соседей:



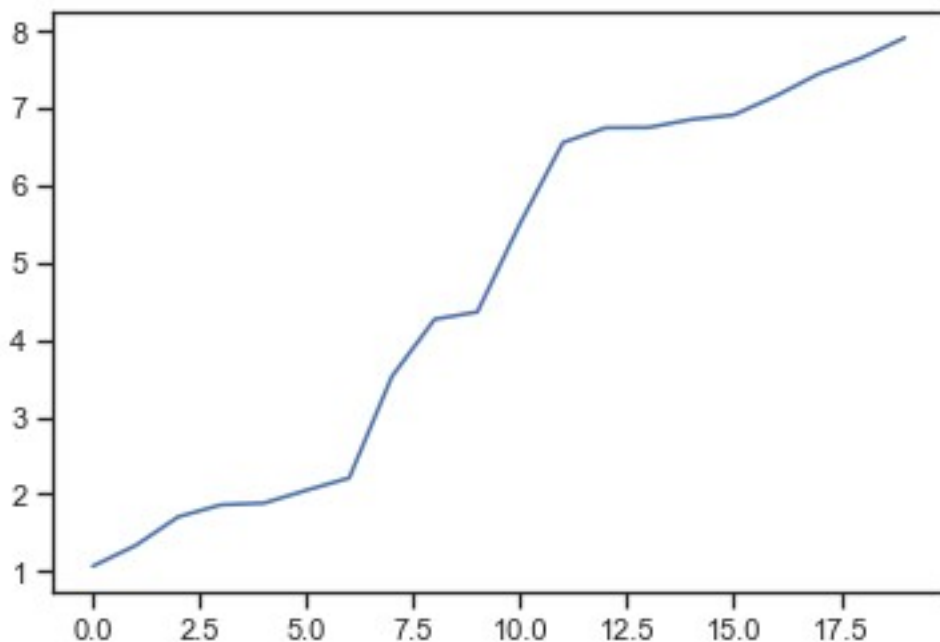
Классы, соответствующие K ближайшим соседям:
 класс=B, количество элементов=2
 класс=C, количество элементов=1

Проверяемая точка: [46.94, 46.13]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=C, расстояние=1.06
 X1=45.94, X2=45.25, y=B, расстояние=1.33
 X1=45.94, X2=44.75, y=B, расстояние=1.70
 X1=45.63, X2=44.81, y=B, расстояние=1.86
 X1=46.25, X2=44.38, y=C, расстояние=1.88
 X1=46.0, X2=44.31, y=C, расстояние=2.05
 X1=45.69, X2=44.31, y=B, расстояние=2.21
 X1=45.0, X2=43.19, y=B, расстояние=3.52
 X1=48.0, X2=42.0, y=C, расстояние=4.26
 X1=45.38, X2=42.06, y=B, расстояние=4.36
 X1=43.44, X2=41.88, y=B, расстояние=5.51
 X1=42.94, X2=40.94, y=A, расстояние=6.55
 X1=44.0, X2=40.06, y=B, расстояние=6.74
 X1=43.63, X2=40.25, y=B, расстояние=6.75
 X1=50.0, X2=40.0, y=D, расстояние=6.85
 X1=42.31, X2=41.0, y=A, расстояние=6.91
 X1=42.44, X2=40.56, y=A, расстояние=7.16
 X1=43.0, X2=39.81, y=B, расстояние=7.45
 X1=41.5, X2=40.75, y=A, расстояние=7.65
 X1=41.94, X2=40.0, y=A, расстояние=7.91

Вывод расстояния для отсортированного списка соседей:



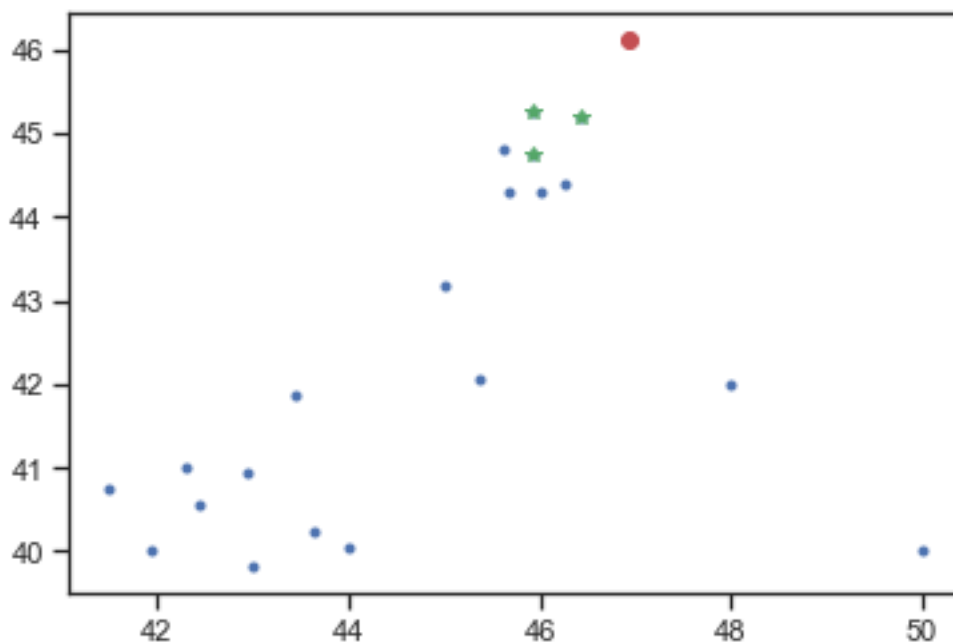
Вывод K ближайших соседей:

X1=46.44, X2=45.19, y=C, расстояние=1.06

X1=45.94, X2=45.25, y=B, расстояние=1.33

X1=45.94, X2=44.75, y=B, расстояние=1.70

Визуализация K ближайших соседей:



Классы, соответствующие K ближайшим соседям:

класс=B, количество элементов=2

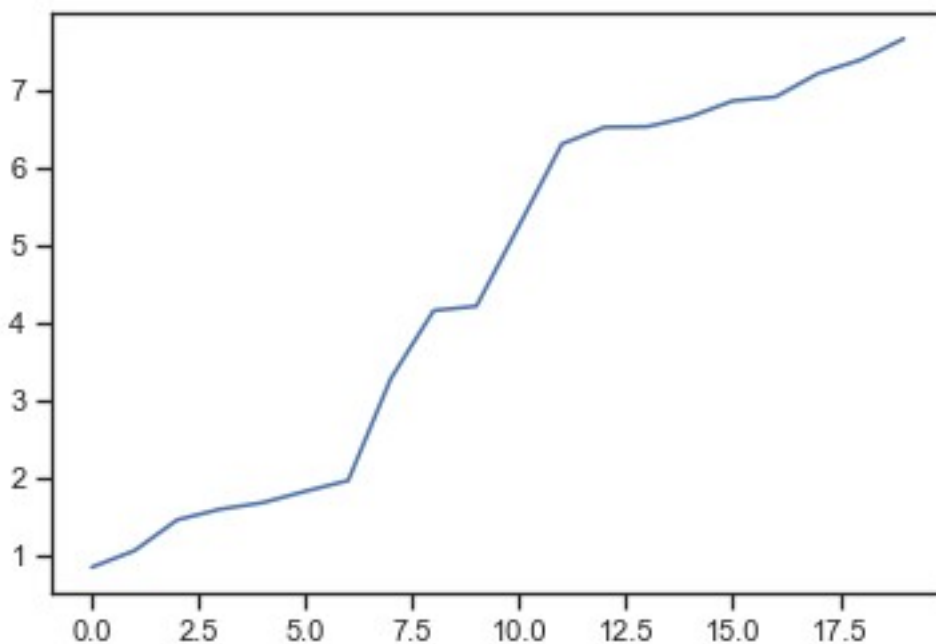
класс=C, количество элементов=1

Проверяемая точка: [46.69, 46.0]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=C, расстояние=0.85
X1=45.94, X2=45.25, y=B, расстояние=1.06
X1=45.94, X2=44.75, y=B, расстояние=1.46
X1=45.63, X2=44.81, y=B, расстояние=1.59
X1=46.25, X2=44.38, y=C, расстояние=1.68
X1=46.0, X2=44.31, y=C, расстояние=1.83
X1=45.69, X2=44.31, y=B, расстояние=1.96
X1=45.0, X2=43.19, y=B, расстояние=3.28
X1=45.38, X2=42.06, y=B, расстояние=4.15
X1=48.0, X2=42.0, y=C, расстояние=4.21
X1=43.44, X2=41.88, y=B, расстояние=5.25
X1=42.94, X2=40.94, y=A, расстояние=6.30
X1=43.63, X2=40.25, y=B, расстояние=6.51
X1=44.0, X2=40.06, y=B, расстояние=6.52
X1=42.31, X2=41.0, y=A, расстояние=6.65
X1=50.0, X2=40.0, y=D, расстояние=6.85
X1=42.44, X2=40.56, y=A, расстояние=6.90
X1=43.0, X2=39.81, y=B, расстояние=7.21
X1=41.5, X2=40.75, y=A, расстояние=7.38
X1=41.94, X2=40.0, y=A, расстояние=7.65

Вывод расстояния для отсортированного списка соседей:

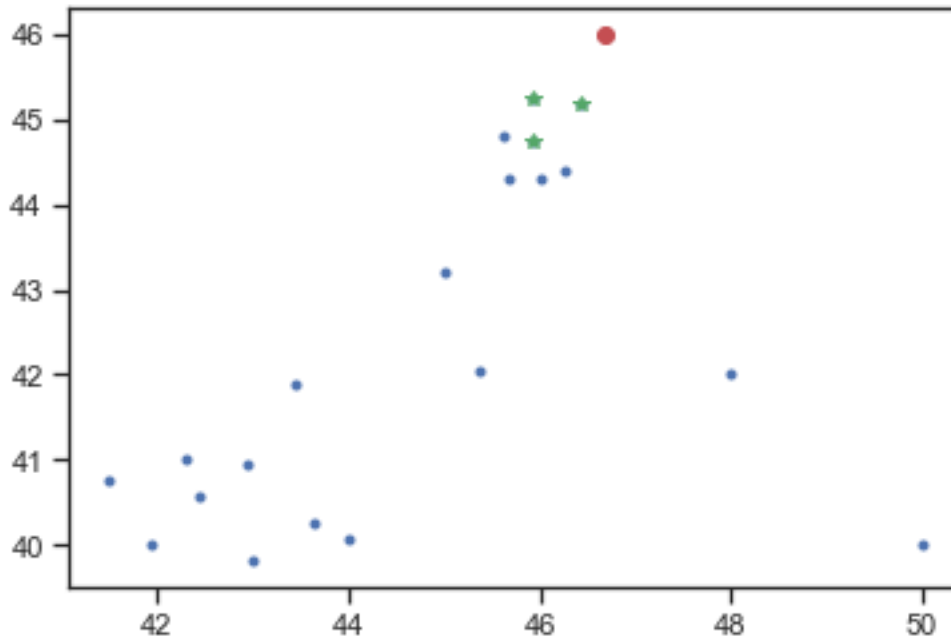


Вывод K ближайших соседей:

X1=46.44, X2=45.19, y=C, расстояние=0.85

X1=45.94, X2=45.25, y=B, расстояние=1.06
X1=45.94, X2=44.75, y=B, расстояние=1.46

Визуализация K ближайших соседей:



Классы, соответствующие K ближайшим соседям:
класс=B, количество элементов=2
класс=C, количество элементов=1

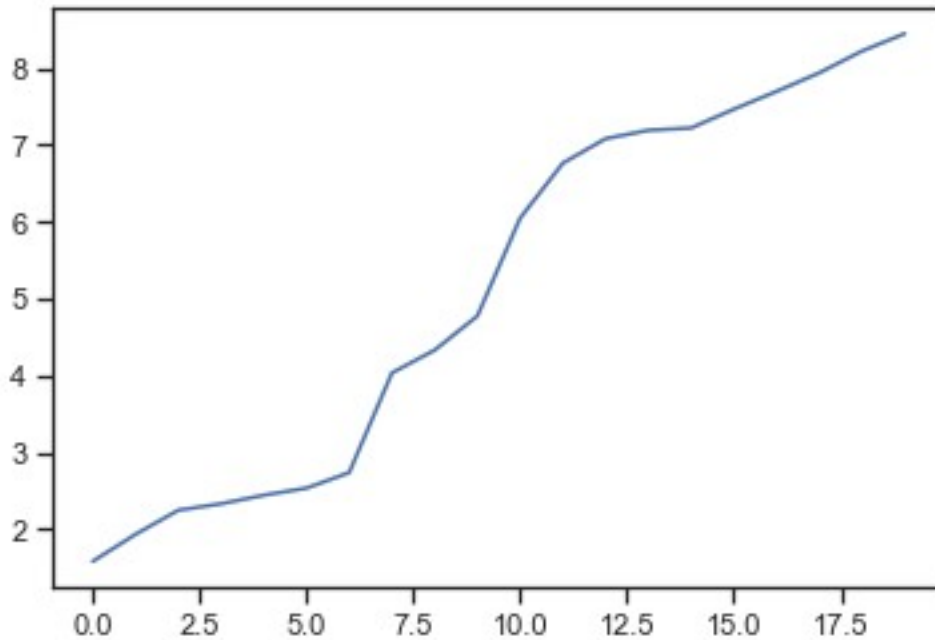
Проверяемая точка: [47.56, 46.31]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=C, расстояние=1.58
X1=45.94, X2=45.25, y=B, расстояние=1.94
X1=45.94, X2=44.75, y=B, расстояние=2.25
X1=46.25, X2=44.38, y=C, расстояние=2.33
X1=45.63, X2=44.81, y=B, расстояние=2.44
X1=46.0, X2=44.31, y=C, расстояние=2.54
X1=45.69, X2=44.31, y=B, расстояние=2.74
X1=45.0, X2=43.19, y=B, расстояние=4.04
X1=48.0, X2=42.0, y=C, расстояние=4.33
X1=45.38, X2=42.06, y=B, расстояние=4.78
X1=43.44, X2=41.88, y=B, расстояние=6.05
X1=50.0, X2=40.0, y=D, расстояние=6.77
X1=42.94, X2=40.94, y=A, расстояние=7.08
X1=44.0, X2=40.06, y=B, расстояние=7.19
X1=43.63, X2=40.25, y=B, расстояние=7.22
X1=42.31, X2=41.0, y=A, расстояние=7.47
X1=42.44, X2=40.56, y=A, расстояние=7.70

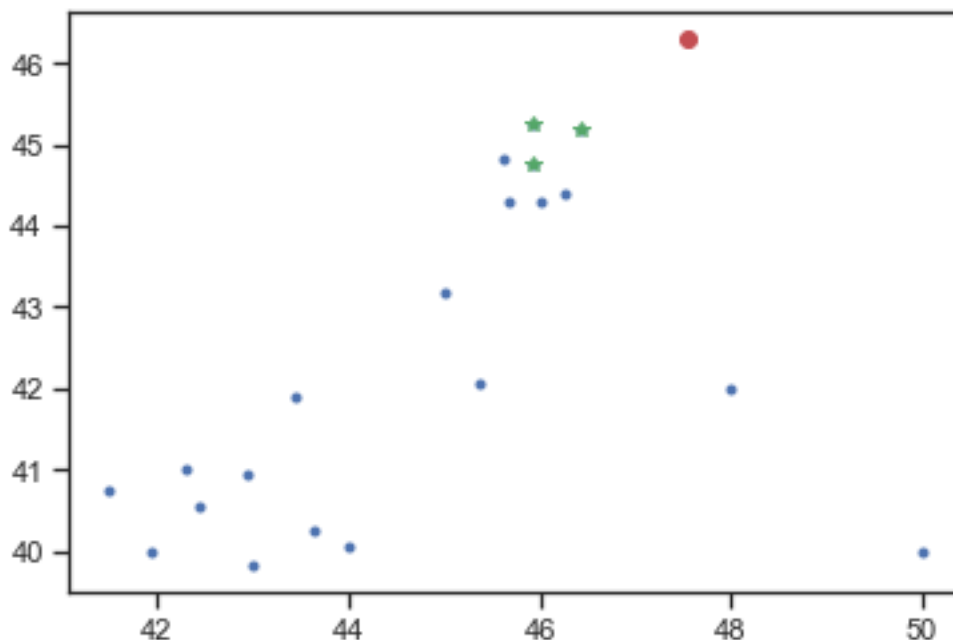
X1=43.0, X2=39.81, y=B, расстояние=7.94
X1=41.5, X2=40.75, y=A, расстояние=8.22
X1=41.94, X2=40.0, y=A, расстояние=8.45

Вывод расстояния для отсортированного списка соседей:



Вывод K ближайших соседей:
X1=46.44, X2=45.19, y=C, расстояние=1.58
X1=45.94, X2=45.25, y=B, расстояние=1.94
X1=45.94, X2=44.75, y=B, расстояние=2.25

Визуализация K ближайших соседей:



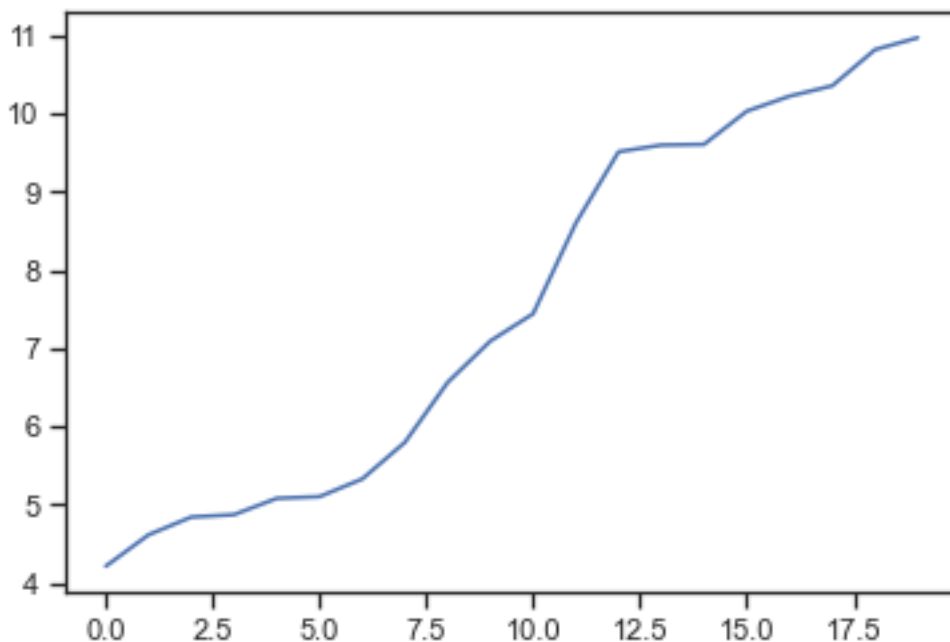
Классы, соответствующие K ближайшим соседям:
 класс=B, количество элементов=2
 класс=C, количество элементов=1

Проверяемая точка: [50.0, 47.44]

Вывод отсортированного списка соседей:

X1=46.44, X2=45.19, y=C, расстояние=4.21
 X1=45.94, X2=45.25, y=B, расстояние=4.61
 X1=46.25, X2=44.38, y=C, расстояние=4.84
 X1=45.94, X2=44.75, y=B, расстояние=4.87
 X1=46.0, X2=44.31, y=C, расстояние=5.08
 X1=45.63, X2=44.81, y=B, расстояние=5.10
 X1=45.69, X2=44.31, y=B, расстояние=5.33
 X1=48.0, X2=42.0, y=C, расстояние=5.80
 X1=45.0, X2=43.19, y=B, расстояние=6.56
 X1=45.38, X2=42.06, y=B, расстояние=7.09
 X1=50.0, X2=40.0, y=D, расстояние=7.44
 X1=43.44, X2=41.88, y=B, расстояние=8.60
 X1=44.0, X2=40.06, y=B, расстояние=9.51
 X1=42.94, X2=40.94, y=A, расстояние=9.60
 X1=43.63, X2=40.25, y=B, расстояние=9.61
 X1=42.31, X2=41.0, y=A, расстояние=10.03
 X1=42.44, X2=40.56, y=A, расстояние=10.22
 X1=43.0, X2=39.81, y=B, расстояние=10.35
 X1=41.5, X2=40.75, y=A, расстояние=10.82
 X1=41.94, X2=40.0, y=A, расстояние=10.97

Вывод расстояния для отсортированного списка соседей:



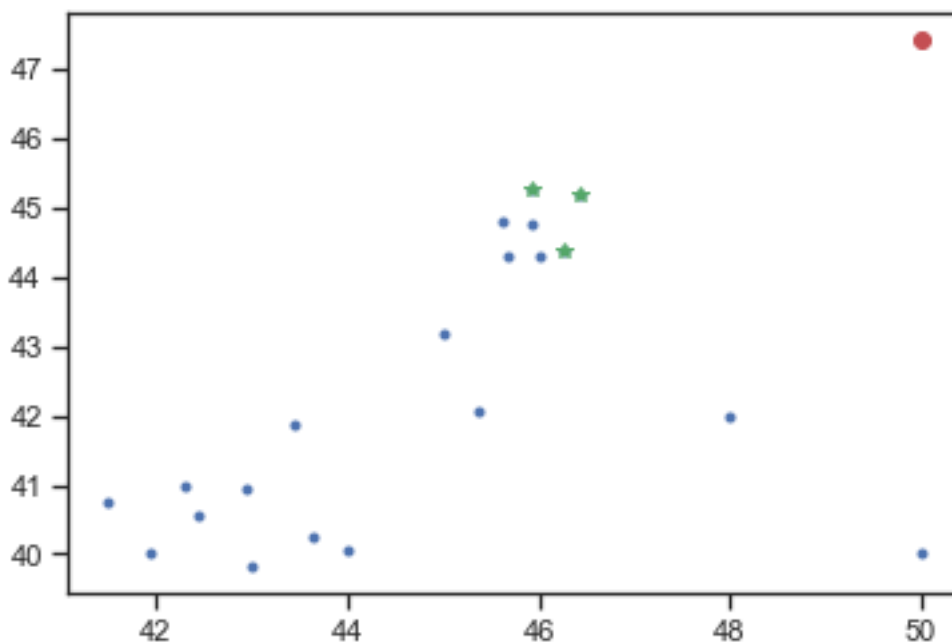
Вывод K ближайших соседей:

X1=46.44, X2=45.19, y=C, расстояние=4.21

X1=45.94, X2=45.25, y=B, расстояние=4.61

X1=46.25, X2=44.38, y=C, расстояние=4.84

Визуализация K ближайших соседей:



Классы, соответствующие K ближайшим соседям:

класс=C, количество элементов=2

класс=B, количество элементов=1

```

['B', 'B', 'B', 'B', 'C']

from sklearn.neighbors import KNeighborsRegressor,
KNeighborsClassifier

KNeighborsClassifierObj = KNeighborsClassifier(n_neighbors=3)
KNeighborsClassifierObj

KNeighborsClassifier(n_neighbors=3)

KNeighborsClassifierObj.fit(data_train[['high', 'low']],
data_train['y_clas'])
KNeighborsClassifierObj.predict(data_test[['high', 'low']])

array(['B', 'B', 'B', 'B', 'C'], dtype=object)

simple_knn_clas = SimpleKNN()
simple_knn_clas.fit(data_train[['high', 'low']], data_train['y_clas'])
simple_knn_clas_prediction = simple_knn_clas.predict(K=3, \

prediction_type=PredictionType.CLASSIFICATION, \
X_test=data_test[['high', 'low']], verbose =
False)
np.array(simple_knn_clas_prediction)

array(['B', 'B', 'B', 'B', 'C'], dtype='<U1')

KNeighborsRegressorObj = KNeighborsRegressor()
KNeighborsRegressorObj

KNeighborsRegressor()

KNeighborsRegressorObj.fit(data_train[['high', 'low']],
data_train['close'])
KNeighborsRegressorObj.predict(data_test[['high', 'low']])

array([45.412, 45.4 , 45.4 , 45.4 , 45.412])

simple_knn_regr = SimpleKNN()
simple_knn_regr.fit(data_train[['high', 'low']], data_train['close'])
simple_knn_regr_prediction = simple_knn_regr.predict(K=5, \
prediction_type=PredictionType.REGRESSION, \
X_test=data_test[['high', 'low']], verbose =
False)
np.array(simple_knn_regr_prediction)

array([45.412, 45.4 , 45.4 , 45.4 , 45.412])

import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from sklearn.datasets import load_iris, load_boston

```


500501	AINV	2012-07-18	7.97	8.035	7.89	7.93	1699100.0
500502	AINV	2012-07-19	7.98	8.000	7.87	7.94	1484300.0
500503	AINV	2012-07-20	7.90	7.930	7.76	7.82	1867200.0
500504	AINV	2012-07-23	7.73	7.820	7.59	7.79	2281600.0
500505	AINV	2012-07-24	7.80	7.820	7.55	7.62	1394000.0

	ex-dividend	split_ratio	adj_open	adj_high	adj_low
adj_close \					
0	0.0	1.0	31.041951	34.112034	27.289627
30.018590					
1	0.0	1.0	29.295415	29.336350	27.160002
27.548879					
2	0.0	1.0	28.183363	30.018590	27.330562
30.018590					
3	0.0	1.0	28.995229	29.766161	27.460188
27.460188					
4	0.0	1.0	27.378319	28.613174	27.289627
28.012803					
...
...					
500501	0.0	1.0	4.563940	4.601161	4.518128
4.541034					
500502	0.0	1.0	4.569666	4.581119	4.506676
4.546760					
500503	0.0	1.0	4.523855	4.541034	4.443685
4.478044					
500504	0.0	1.0	4.426506	4.478044	4.346336
4.460864					
500505	0.0	1.0	4.466591	4.478044	4.323431
4.363516					

	adj_volume	y_clas
0	44739900.0	2
1	10897100.0	1
2	4705200.0	2
3	4274400.0	1
4	3464400.0	1
...
500501	1699100.0	1
500502	1484300.0	1
500503	1867200.0	1
500504	2281600.0	1
500505	1394000.0	1

[500506 rows x 15 columns]

```
# И выведем его статистические характеристики
df.describe()
# Для обучения моделей не обязательно создавать DataFrame
# можно использовать массивы numpy
```

	a	b
count	500506.000000	500506.000000
mean	32.004692	31.096170
std	39.937695	38.778618
min	0.015630	0.015630
25%	13.600000	13.100000
50%	24.420000	23.670000
75%	39.750000	38.650000
max	1209.000000	1166.000000

Разделение выборки на обучающую и тестовую

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.5,
                                                    random_state=1)
```

Размер обучающей выборки

iris_X_train.shape, iris_y_train.shape

```
X_train.shape, len(y_train)
```

```
((250253, 2), 250253)
```

Размер тестовой выборки

iris_X_test.shape, iris_y_test.shape

```
X_test.shape, len(y_test)
```

```
((250253, 2), 250253)
```

```
np.unique(y_train)
```

```
array([1, 2, 3])
```

```
np.unique(y_test)
```

```
array([1, 2, 3])
```

```
def class_proportions(array: np.ndarray) -> Dict[int, Tuple[int,
float]]:
```

```
    """
```

Вычисляет пропорции классов

array - массив, содержащий метки классов

```
    """
```

Получение меток классов и количества меток каждого класса

```
labels, counts = np.unique(array, return_counts=True)
```

Превращаем количество меток в процент их встречаемости

делим количество меток каждого класса на общее количество меток

```
counts_perc = counts/array.size
```

Теперь sum(counts_perc)==1.0

Создаем результирующий словарь,

ключом словаря является метка класса,

а значением словаря процент встречаемости метки

```
res = dict()
```

```
for label, count2 in zip(labels, zip(counts, counts_perc)):
```

```

        res[label] = count2
    return res

def print_class_proportions(array: np.ndarray):
    """
    Вывод пропорций классов
    """
    proportions = class_proportions(array)
    if len(proportions)>0:
        print('Метка \t Количество \t Процент встречаемости')
    for i in proportions:
        val, val_perc = proportions[i]
        val_perc_100 = round(val_perc * 100, 2)
        print('{} \t {} \t \t {}%'.format(i, val, val_perc_100))

# Для обучающей выборки
Counter(y_train)

Counter({1: 198436, 2: 42818, 3: 8999})

# Для тестовой выборки
Counter(y_test)

Counter({2: 42779, 1: 198303, 3: 9171})

# 2 ближайших соседа
cl1_1 = KNeighborsClassifier(n_neighbors=2)
cl1_1.fit(X_train, y_train)
target1_1 = cl1_1.predict(X_test)
len(target1_1), target1_1

(250253, array([2, 1, 1, ..., 1, 1, 1]))

# 10 ближайших соседей
cl1_2 = KNeighborsClassifier(n_neighbors=10)
cl1_2.fit(X_train, y_train)
target1_2 = cl1_2.predict(X_test)
len(target1_2), target1_2

(250253, array([2, 1, 1, ..., 1, 1, 1]))

# iris_y_test - эталонное значение классов из исходной (тестовой)
выборки
# target* - предсказанное значение классов

# 2 ближайших соседа
accuracy_score(y_test, target1_1)

0.9946933703092471

def accuracy_score_for_classes(
    y_true: np.ndarray,

```

```

y_pred: np.ndarray) -> Dict[int, float]:
"""
Вычисление метрики ассигасу для каждого класса
y_true - истинные значения классов
y_pred - предсказанные значения классов
Возвращает словарь: ключ - метка класса,
значение - Ассигасу для данного класса
"""

# Для удобства фильтрации сформируем Pandas DataFrame
d = {'t': y_true, 'p': y_pred}
df = pd.DataFrame(data=d)
# Метки классов
classes = np.unique(y_true)
# Результирующий словарь
res = dict()
# Перебор меток классов
for c in classes:
    # отфильтруем данные, которые соответствуют
    # текущей метке класса в истинных значениях
    temp_dataflt = df[df['t']==c]
    # расчет ассигасу для заданной метки класса
    temp_acc = accuracy_score(
        temp_dataflt['t'].values,
        temp_dataflt['p'].values)
    # сохранение результата в словарь
    res[c] = temp_acc
return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики ассигасу для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))

# 2 ближайших соседа
print_accuracy_score_for_classes(y_test, target1_1)

Метка      Accuracy
1      0.9985275058874551
2      0.9791720236564669
3      0.9841892923345328

# 10 ближайших соседей
print_accuracy_score_for_classes(y_test, target1_2)

```

Метка	Accuracy
1	0.9977357881625594
2	0.9873769840342224
3	0.989859339221459

Конвертация целевого признака в бинарный

```
def convert_target_to_binary(array:np.ndarray, target:int) ->
np.ndarray:
```

```
    # Если целевой признак совпадает с указанным, то 1 иначе 0
    res = [1 if x==target else 0 for x in array]
    return res
```

Если целевой признак ==2,

то будем считать этот случай 1 в бинарном признаке

```
bin_iris_y_train = convert_target_to_binary(y_train, 2)
list(zip(y_train, bin_iris_y_train))[:10]
```

```
[(1, 0),
 (1, 0),
 (2, 1),
 (1, 0),
 (1, 0),
 (1, 0),
 (2, 1),
 (2, 1),
 (2, 1),
 (1, 0)]
```

```
bin_iris_y_test = convert_target_to_binary(y_test, 2)
list(zip(y_test, bin_iris_y_test))[:10]
```

```
[(2, 1),
 (1, 0),
 (1, 0),
 (1, 0),
 (1, 0),
 (1, 0),
 (1, 0),
 (1, 0),
 (2, 1),
 (1, 0)]
```

Конвертация предсказанных признаков

```
bin_target1_1 = convert_target_to_binary(target1_1, 2)
bin_target1_2 = convert_target_to_binary(target1_2, 2)
```

```
balanced_accuracy_score(bin_iris_y_test, bin_target1_1)
```

```
0.9885328678198275
```

```
balanced_accuracy_score(bin_iris_y_test, bin_target1_2)
```



```
0.9923823042538251
```

```
confusion_matrix(bin_iris_y_test, bin_target1_1, labels=[0, 1])
```

```
array([[207037,    437],  
       [   891,  41888]], dtype=int64)
```

```
tn, fp, fn, tp = confusion_matrix(bin_iris_y_test,  
bin_target1_1).ravel()  
tn, fp, fn, tp
```

```
(207037, 437, 891, 41888)
```

```
# Пример для небинарной классификации
```

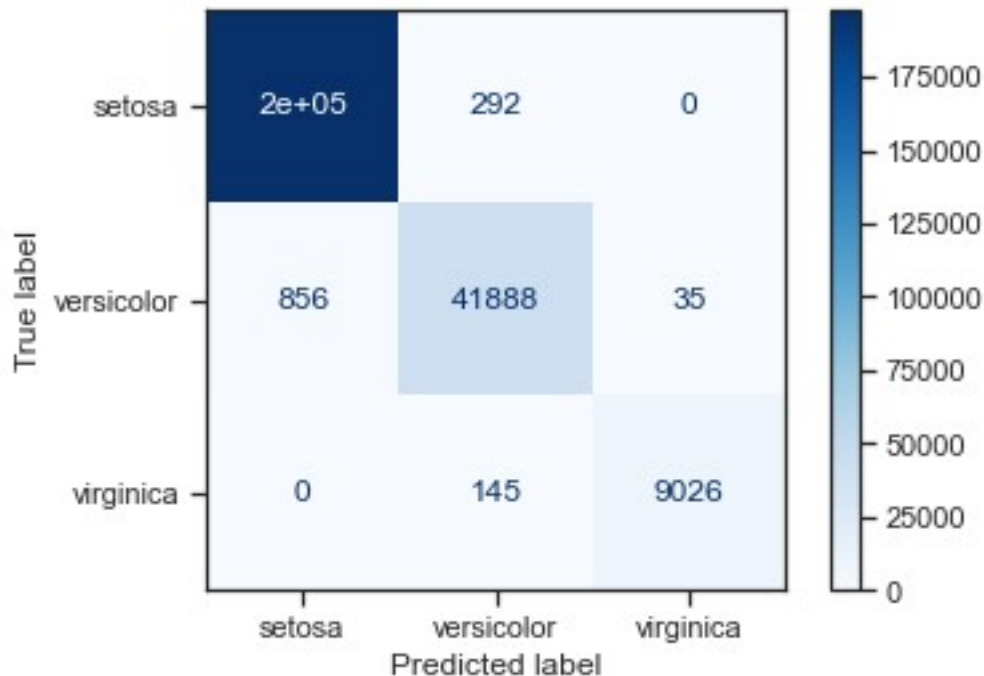
```
confusion_matrix(y_test, target1_1, labels=[0, 1, 2])
```

```
array([[    0,     0,     0],  
       [    0, 198011,   292],  
       [    0,    856,  41888]], dtype=int64)
```

```
plot_confusion_matrix(clf_1, X_test, y_test,  
                      display_labels=iris.target_names,  
                      cmap=plt.cm.Blues)
```

```
C:\Users\Админ\AppData\Local\Packages\  
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-  
packages\Python39\site-packages\sklearn\utils\deprecation.py:87:  
FutureWarning: Function plot_confusion_matrix is deprecated; Function  
'plot_confusion_matrix' is deprecated in 1.0 and will be removed in  
1.2. Use one of the class methods:  
ConfusionMatrixDisplay.from_predictions or  
ConfusionMatrixDisplay.from_estimator.  
  warnings.warn(msg, category=FutureWarning)
```

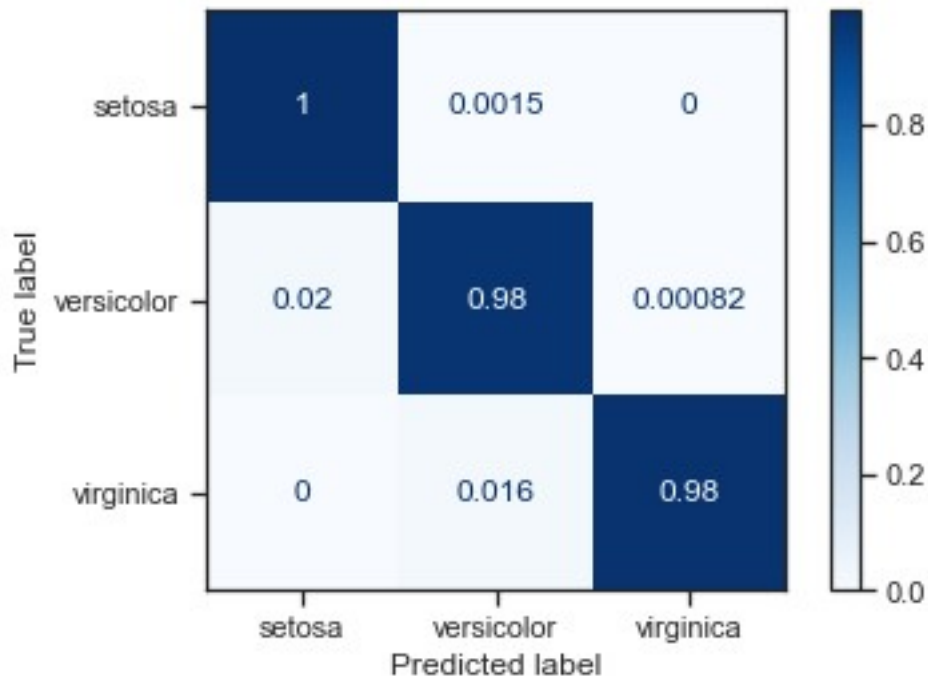
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at  
0x2931cac5df0>
```



```
plot_confusion_matrix(clf_1, X_test, y_test,
                      display_labels=iris.target_names,
                      cmap=plt.cm.Blues, normalize='true')
```

C:\Users\Админ\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-packages\Python39\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
warnings.warn(msg, category=FutureWarning)

<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2931cac5f10>



```
fig, ax = plt.subplots(1, 2, sharex='col', sharey='row',
figsize=(15,5))
```

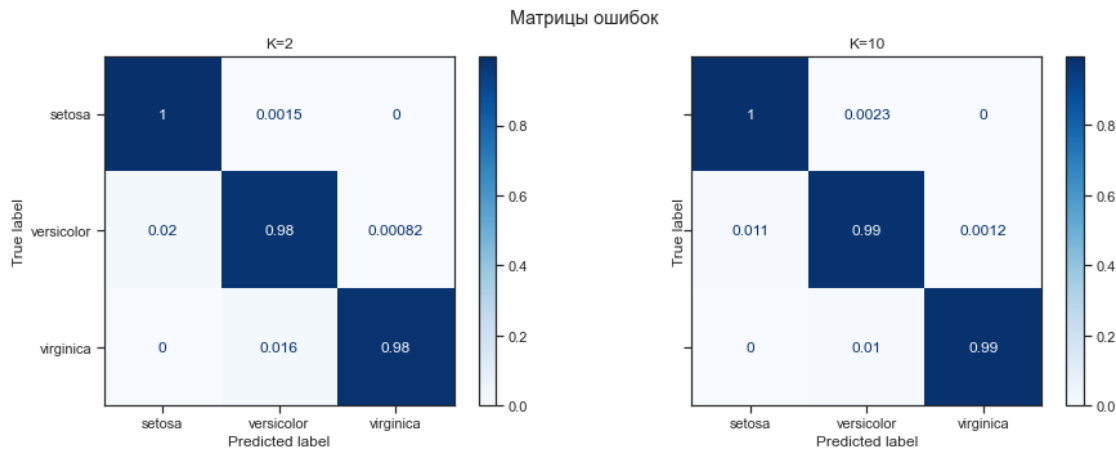
```
plot_confusion_matrix(clf_1, X_test, y_test,
display_labels=iris.target_names,
cmap=plt.cm.Blues, normalize='true', ax=ax[0])
```

```
plot_confusion_matrix(clf_2, X_test, y_test,
display_labels=iris.target_names,
cmap=plt.cm.Blues, normalize='true', ax=ax[1])
```

```
fig.suptitle('Матрицы ошибок')
ax[0].title.set_text('K=2')
ax[1].title.set_text('K=10')
```

```
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in
1.2. Use one of the class methods:
ConfusionMatrixDisplay.from_predictions or
ConfusionMatrixDisplay.from_estimator.
warnings.warn(msg, category=FutureWarning)
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\utils\deprecation.py:87:
FutureWarning: Function plot_confusion_matrix is deprecated; Function
`plot_confusion_matrix` is deprecated in 1.0 and will be removed in
```

1.2. Use one of the class methods:
 ConfusionMatrixDisplay.from_predictions or
 ConfusionMatrixDisplay.from_estimator.
 warnings.warn(msg, category=FutureWarning)



По умолчанию метрики считаются для 1 класса бинарной классификации
Для 2 ближайших соседей

```
precision_score(bin_iris_y_test, bin_target1_1),  
recall_score(bin_iris_y_test, bin_target1_1)
```

(0.9896751329001772, 0.9791720236564669)

Для 10 ближайших соседей

```
precision_score(bin_iris_y_test, bin_target1_2),  
recall_score(bin_iris_y_test, bin_target1_2)
```

(0.9873308244314065, 0.9873769840342224)

Параметры TP, TN, FP, FN считаются как сумма по всем классам
precision_score(y_test, target1_1, average='micro')

0.9946933703092471

Параметры TP, TN, FP, FN считаются отдельно для каждого класса
и берется среднее значение, дисбаланс классов не учитывается.

```
precision_score(y_test, target1_1, average='macro')
```

0.9938360134175218

Параметры TP, TN, FP, FN считаются отдельно для каждого класса
и берется средневзвешенное значение, дисбаланс классов учитывается
в виде веса классов (вес - количество истинных значений каждого класса).

```
precision_score(y_test, target1_1, average='weighted')
```

0.9946826423067406

```
f1_score(bin_iris_y_test, bin_target1_2)
```

```

0.9873539036933147
f1_score(y_test, target1_1, average='micro')
0.9946933703092471
f1_score(y_test, target1_1, average='macro')
0.9905441206452729
f1_score(y_test, target1_1, average='weighted')
0.9946803031267678
classification_report(y_test, target1_1,
                      target_names=iris.target_names,
                      output_dict=True)
{'setosa': {'precision': 0.9956956156627293,
            'recall': 0.9985275058874551,
            'f1-score': 0.9971095500667221,
            'support': 198303},
 'versicolor': {'precision': 0.9896751329001772,
                 'recall': 0.9791720236564669,
                 'f1-score': 0.9843955630757661,
                 'support': 42779},
 'virginica': {'precision': 0.996137291689659,
               'recall': 0.9841892923345328,
               'f1-score': 0.9901272487933304,
               'support': 9171},
 'accuracy': 0.9946933703092471,
 'macro avg': {'precision': 0.9938360134175218,
               'recall': 0.9872962739594849,
               'f1-score': 0.9905441206452729,
               'support': 250253},
 'weighted avg': {'precision': 0.9946826423067406,
                  'recall': 0.9946933703092471,
                  'f1-score': 0.9946803031267678,
                  'support': 250253}}

```

```

# Обучим модели на задаче бинарной классификации,
# чтобы получить вероятности классов

```

```

# 2 ближайших соседа
bin_clf_1 = KNeighborsClassifier(n_neighbors=2)
bin_clf_1.fit(X_train, bin_iris_y_train)
# предскажем метки классов
bin_clf_1.predict(X_test)

array([1, 0, 0, ..., 0, 0, 0])

```

```

# Классы возвращаются в следующем порядке
bin_cl1_1.classes_

array([0, 1])

# предскажем вероятности классов
proba_target1_1 = bin_cl1_1.predict_proba(X_test)
len(proba_target1_1), proba_target1_1

(250253,
 array([[0., 1.],
        [1., 0.],
        [1., 0.],
        ...,
        [1., 0.],
        [1., 0.],
        [1., 0.]])

# вероятность единичного (истинного) класса
true_proba_target1_1 = proba_target1_1[:,1]
true_proba_target1_1

array([1., 0., 0., ..., 0., 0., 0.])

fpr, tpr, thresholds = roc_curve(bin_iris_y_test,
true_proba_target1_1,
                                pos_label=1)

fpr, tpr, thresholds

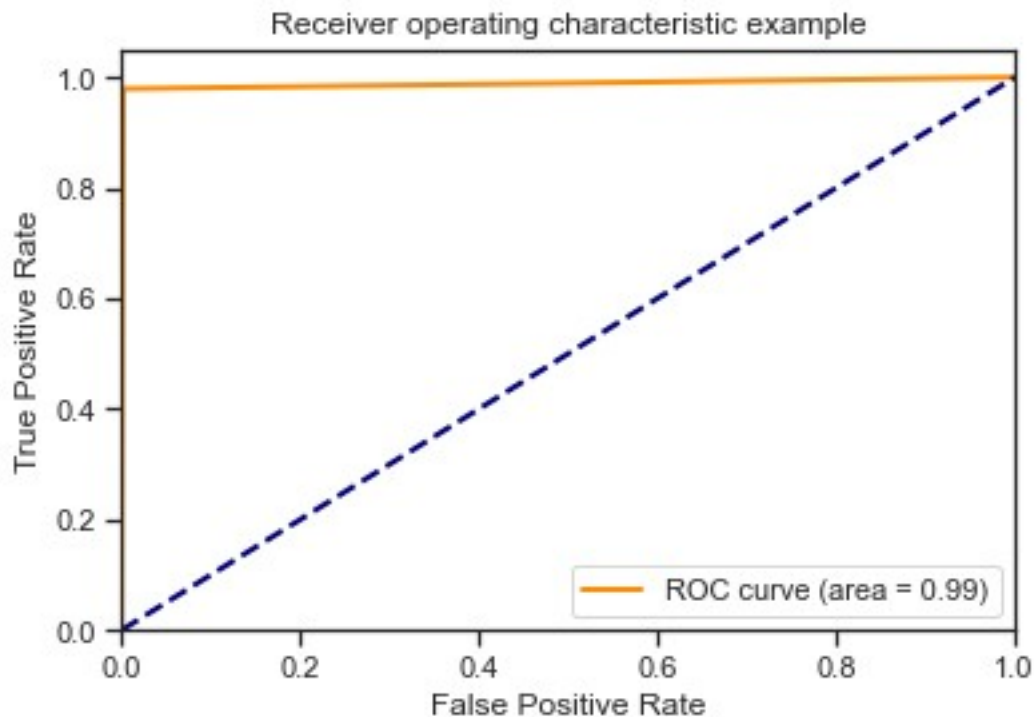
(array([0.          , 0.0016243 , 0.00475722, 1.          ],
 array([0.          , 0.97692793, 0.9932911 , 1.          ],
 array([2. , 1. , 0.5, 0. ]))

# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, pos_label, average):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

```

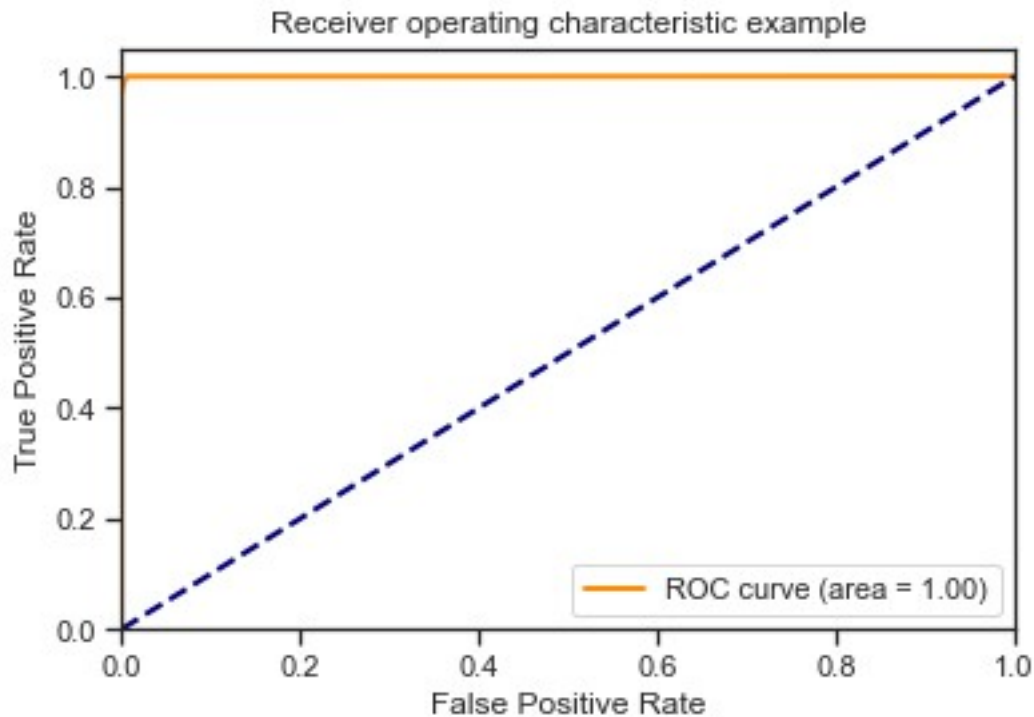
```
# Для 2 ближайших соседей
```

```
draw_roc_curve(bin_iris_y_test, bin_target1_1, pos_label=1,  
average='micro')
```



```
# Для 10 ближайших соседей
```

```
bin_cl1_2 = KNeighborsClassifier(n_neighbors=10)  
bin_cl1_2.fit(X_train, bin_iris_y_train)  
proba_target2_1 = bin_cl1_2.predict_proba(X_test)  
true_proba_target2_1 = proba_target2_1[:,1]  
roc_curve_k10_res = roc_curve(bin_iris_y_test, true_proba_target2_1,  
pos_label=1)  
roc_curve_k10_res  
  
(array([0.00000000e+00, 9.15777399e-05, 3.08472387e-04, 8.91677993e-  
04,  
        1.69659813e-03, 2.51597791e-03, 3.41247578e-03, 4.40537128e-  
03,  
        5.30186915e-03, 6.20800679e-03, 7.35513848e-03,  
        1.00000000e+00]),  
 array([0.          , 0.96182706, 0.96736717, 0.97543187, 0.98157975,  
        0.98683934, 0.99060287, 0.99415601, 0.99691437, 0.99866757,  
        0.99955586, 1.          ]),  
 array([2. , 1. , 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0. ]))  
  
draw_roc_curve(bin_iris_y_test, true_proba_target2_1, pos_label=1,  
average='micro')
```



```
def simple_roc_curve(y_true: np.ndarray, y_score: np.ndarray):
    """
    Простая реализация построения ROC-кривой
    """
    # Результирующие массивы
    tpr_arr = []
    fpr_arr = []
    # Получаем уникальные значения вероятностей
    unique_scores = np.unique(y_score)
    # и сортируем их в обратном порядке
    unique_scores_sorted = np.sort(unique_scores)[::-1]

    # Считаем количество истинных 0 и 1 значений
    # Истинные 1 - это TP+FN
    P = np.sum(y_true > 0)
    # Истинные 0 - это FP+TN
    N = y_true.size - P

    # Внешний цикл по уникальным значениям вероятностей
    for t_cur in unique_scores_sorted:
        FP, TP = 0, 0
        # Вложенный цикл по всем предсказаниям
        for y, score in zip(y_true, y_score):
            # Если вероятность текущего предсказания больше пороговой
            if score >= t_cur:
                # и истинное значение = 1
                if y > 0:
```



```

        TP = TP + 1
    else:
        FP = FP + 1

    # Вычисление значений TPR и FPR для текущего порога
    вероятности
    TPR = TP/P
    FPR = FP/N
    # Добавление их в списки
    tpr_arr.append(TPR)
    fpr_arr.append(FPR)

    # Формат вывода совпадает с roc_curve
    return np.array(fpr_arr), np.array(tpr_arr), unique_scores_sorted

simple_roc_curve(np.array(bin_iris_y_test),
np.array(true_proba_target2_1))

(array([9.15777399e-05, 3.08472387e-04, 8.91677993e-04, 1.69659813e-
03,
        2.51597791e-03, 3.41247578e-03, 4.40537128e-03, 5.30186915e-
03,
        6.20800679e-03, 7.35513848e-03, 1.00000000e+00]),
array([0.96182706, 0.96736717, 0.97543187, 0.98157975, 0.98683934,
        0.99060287, 0.99415601, 0.99691437, 0.99866757, 0.99955586,
        1.          ]),
array([1. , 0.9, 0.8, 0.7, 0.6, 0.5, 0.4, 0.3, 0.2, 0.1, 0. ]))

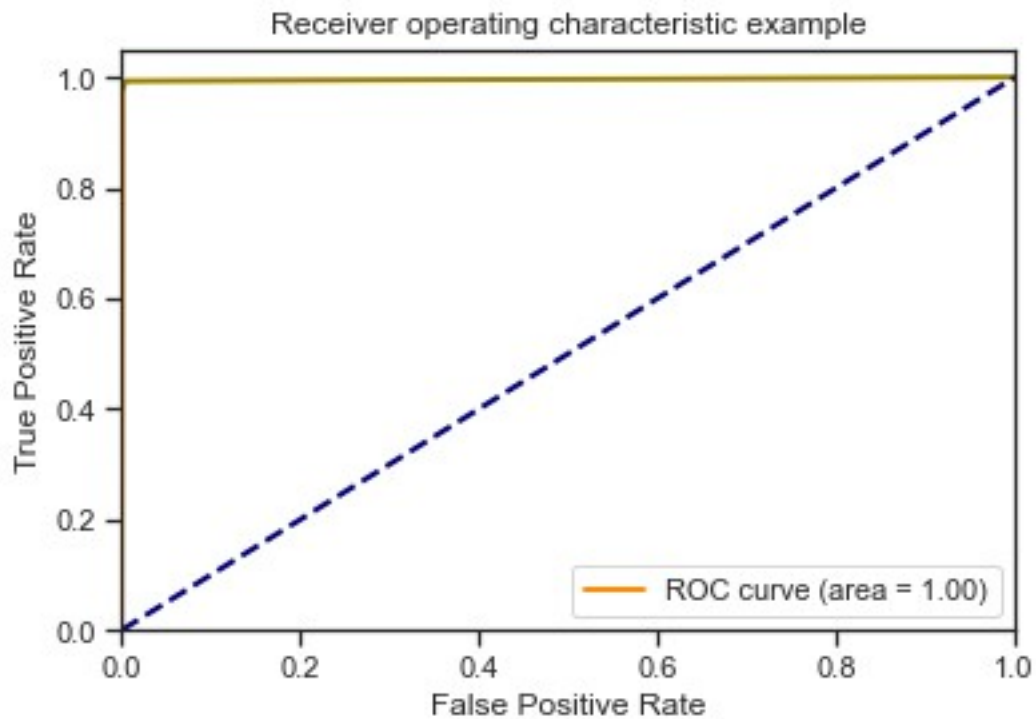
# Отрисовка двух ROC-кривых
def draw_roc_curve_2(y_true, y_score, pos_label, average):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)

    # стандартный метод из sklearn
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    # собственная реализация
    fpr2, tpr2, thresholds2 = simple_roc_curve(y_true, y_score)

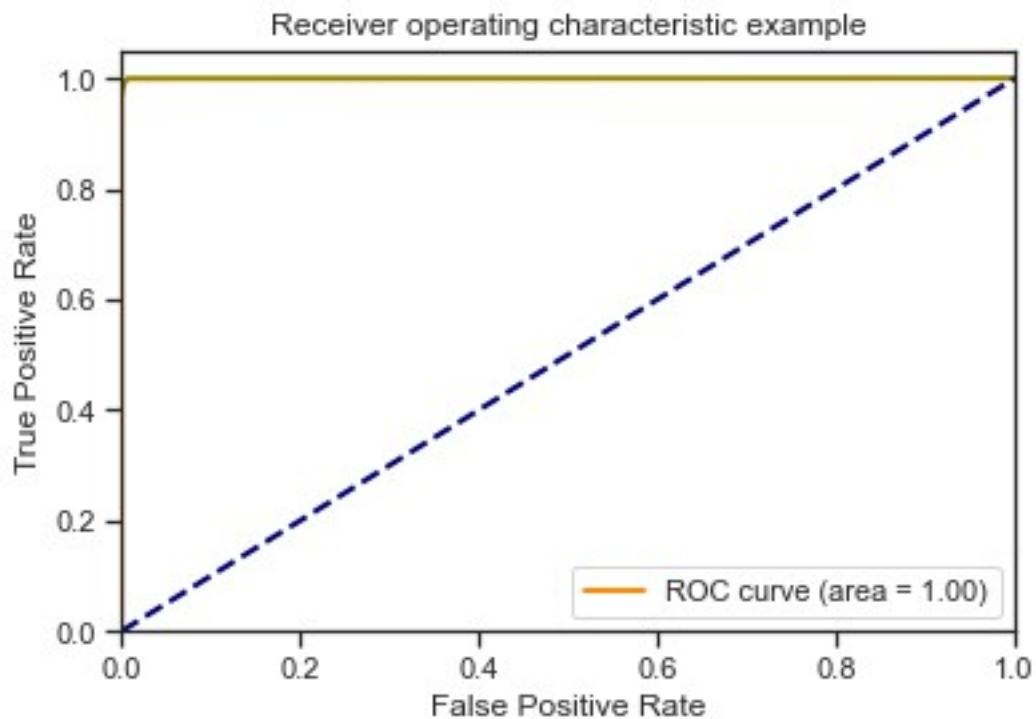
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot(fpr2, tpr2, color='green', alpha=0.5)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

```

```
draw_roc_curve_2(np.array(bin_iris_y_test),  
np.array(true_proba_target1_1), pos_label=1, average='micro')
```



```
draw_roc_curve_2(np.array(bin_iris_y_test),  
np.array(true_proba_target2_1), pos_label=1, average='micro')
```



```

# Для 10 ближайших соседей
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train, bin_iris_y_train)
proba_lr = lr.predict_proba(X_test)
true_proba_lr = proba_lr[:,1]
roc_curve_lr_res = roc_curve(bin_iris_y_test, true_proba_lr,
pos_label=1)
roc_curve_lr_res

(array([0.00000000e+00, 4.81988105e-06, 4.68010450e-03, ...,
        9.99980720e-01, 9.99980720e-01, 1.00000000e+00]),
 array([0.          , 0.          , 0.          , ..., 0.99992987, 1.
        ,
        1.          ]),
 array([2.00000000e+00, 1.00000000e+00, 9.89203685e-01, ...,
        3.95179954e-03, 9.51726033e-04, 1.42345234e-04]))

from IPython.display import Image
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_iris, load_boston
from sklearn.neighbors import KNeighborsRegressor,
KNeighborsClassifier
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.model_selection import KFold, RepeatedKFold, LeaveOneOut,
LeavePOut, ShuffleSplit, StratifiedKFold
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score,
classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error,
mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.model_selection import learning_curve, validation_curve
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")

data = pd.read_csv('WIKI_PRICES1.csv', sep=",")
data1 = {'a': [], 'b': []}
data2 = {'c': []}
df = pd.DataFrame(data1)
df1 = pd.DataFrame(data2)
iris = load_iris()
df['a'] = data['high']
df['b'] = data['low']
df1['c'] = data['close']

```

```

df = df.astype({'a': float, 'b': float})
df1 = df1.astype({'c': np.int8})
X = df.to_numpy()
y = df1.iloc[:, 0].tolist()

# Разделение выборки на обучающую и тестовую
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.5,
                                                    random_state=1)

# В моделях k-ближайших соседей большое значение k
# ведёт к большому смещению и низкой дисперсии (недообучению)
# 70 ближайших соседей
cl1_1 = KNeighborsClassifier(n_neighbors=70)
cl1_1.fit(X_train, y_train)
target1_0 = cl1_1.predict(X_train)
target1_1 = cl1_1.predict(X_test)
accuracy_score(y_train, target1_0), accuracy_score(y_test, target1_1)

(0.7787119435131646, 0.7702604963776658)

# 5 ближайших соседей
cl1_2 = KNeighborsClassifier(n_neighbors=5)
cl1_2.fit(X_train, y_train)
target1_0 = cl1_2.predict(X_train)
target1_1 = cl1_2.predict(X_test)
accuracy_score(y_train, target1_0), accuracy_score(y_test, target1_1)

(0.8212049406001127, 0.7483226974302006)

# 5 ближайших соседей
cl1_2 = KNeighborsClassifier(n_neighbors=5)
cl1_2.fit(X_train, y_train)
target1_0 = cl1_2.predict(X_train)
target1_1 = cl1_2.predict(X_test)
accuracy_score(y_train, target1_0), accuracy_score(y_test, target1_1)

(0.8212049406001127, 0.7483226974302006)

# 1 ближайший сосед - "условное" переобучение
cl1_2 = KNeighborsClassifier(n_neighbors=1)
cl1_2.fit(X_train, y_train)
target1_0 = cl1_2.predict(X_train)
target1_1 = cl1_2.predict(X_test)
accuracy_score(y_train, target1_0), accuracy_score(y_test, target1_1)

(0.9356051675704188, 0.7249383623772742)

scores = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                          X, y, cv=3)

# Значение метрики ассигасу для 3 фолдов
scores

```

```

array([0.71706946, 0.71726556, 0.73449816])

# Усредненное значение метрики accuracy для 3 фолдов
np.mean(scores)

0.7229443920251398

# использование метрики f1
scores = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                        X, y, cv=3,
                        scoring='f1_weighted')
scores, np.mean(scores)

(array([0.71752624, 0.71676814, 0.73396968]), 0.7227546878077792)

scoring = {'precision': 'precision_weighted',
           'recall': 'recall_weighted',
           'f1': 'f1_weighted'}

scores = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        X, y, scoring=scoring,
                        cv=3, return_train_score=True)

scores

C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))

{'fit_time': array([0.34757686, 0.35644412, 0.36185503]),
 'score_time': array([3.66560793, 4.21687341, 3.71113873]),
 'test_precision': array([0.71869359, 0.71684451, 0.73408341]),
 'train_precision': array([0.83993227, 0.84254582, 0.83841761]),
 'test_recall': array([0.71706946, 0.71726556, 0.73449816]),
 'train_recall': array([0.84003057, 0.84266838, 0.83856553]),
 'test_f1': array([0.71752624, 0.71676814, 0.73396968]),
 'train_f1': array([0.83976023, 0.84241472, 0.83831462])}

# Возвращаются индексы элементов
X1 = ["a", "b", "c"]
kf = KFold(n_splits=3)

```

```

for train, test in kf.split(X1):
    print("%s %s" % (train, test))

[1 2] [0]
[0 2] [1]
[0 1] [2]

X1 = range(12)
kf = KFold(n_splits=3)
for train, test in kf.split(X1):
    print("%s %s" % (train, test))

[ 4  5  6  7  8  9 10 11] [0 1 2 3]
[ 0  1  2  3  8  9 10 11] [4 5 6 7]
[0 1 2 3 4 5 6 7] [ 8  9 10 11]

kf = KFold(n_splits=5)
scores = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                          X, y, scoring='f1_weighted',
                          cv=kf)

scores

array([0.68366378, 0.77166522, 0.69918831, 0.72099076, 0.74398748])

kf = KFold(n_splits=5)
scores = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        X, y, scoring=scoring,
                        cv=kf, return_train_score=True)

scores

C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Recall is ill-defined
and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Админ\AppData\Local\Packages\

```

```

PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Recall is ill-defined
and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Recall is ill-defined
and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Precision is ill-
defined and being set to 0.0 in labels with no predicted samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\metrics\
_classification.py:1318: UndefinedMetricWarning: Recall is ill-defined
and being set to 0.0 in labels with no true samples. Use
`zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))

{'fit_time': array([0.42319226, 0.42117333, 0.42702055, 0.4431479 ,
0.4351244 ]),
'score_time': array([2.26196647, 2.20793629, 2.32695794, 2.19190335,
2.42379904]),
'test_precision': array([0.68647215, 0.77193347, 0.69933414,
0.7211375 , 0.74425084]),
'train_precision': array([0.84359613, 0.83104049, 0.84294598,
0.84130726, 0.83333579]),

```

```

'test_recall': array([0.68266368, 0.77207021, 0.69971329, 0.72150128,
0.74448807]),
'train_recall': array([0.84377529, 0.8312134 , 0.84312134,
0.84149798, 0.83349109]),
'test_f1': array([0.68366378, 0.77166522, 0.69918831, 0.72099076,
0.74398748]),
'train_f1': array([0.84351657, 0.83094216, 0.84286511, 0.84124014,
0.83323878])}

```

```

X1 = range(12)
kf = RepeatedKfold(n_splits=3, n_repeats=2)
for train, test in kf.split(X1):
    print("%s %s" % (train, test))

```

```

[ 2  3  5  6  7  8 10 11] [0 1 4 9]
[ 0  1  4  6  7  8  9 11] [ 2  3  5 10]
[ 0  1  2  3  4  5  9 10] [ 6  7  8 11]
[0 1 2 3 4 5 7 8] [ 6  9 10 11]
[ 1  5  6  7  8  9 10 11] [0 2 3 4]
[ 0  2  3  4  6  9 10 11] [1 5 7 8]

```

```

X1 = range(12)
# Эквивалент KFold(n_splits=n)
kf = LeaveOneOut()
for train, test in kf.split(X1):
    print("%s %s" % (train, test))

```

```

[ 1  2  3  4  5  6  7  8  9 10 11] [0]
[ 0  2  3  4  5  6  7  8  9 10 11] [1]
[ 0  1  3  4  5  6  7  8  9 10 11] [2]
[ 0  1  2  4  5  6  7  8  9 10 11] [3]
[ 0  1  2  3  5  6  7  8  9 10 11] [4]
[ 0  1  2  3  4  6  7  8  9 10 11] [5]
[ 0  1  2  3  4  5  7  8  9 10 11] [6]
[ 0  1  2  3  4  5  6  8  9 10 11] [7]
[ 0  1  2  3  4  5  6  7  9 10 11] [8]
[ 0  1  2  3  4  5  6  7  8 10 11] [9]
[ 0  1  2  3  4  5  6  7  8  9 11] [10]
[ 0  1  2  3  4  5  6  7  8  9 10] [11]

```

```

X1 = range(12)
# Эквивалент KFold(n_splits=n)
kf = LeavePOut(2)
for train, test in kf.split(X1):
    print("%s %s" % (train, test))

```

```

[ 2  3  4  5  6  7  8  9 10 11] [0 1]
[ 1  3  4  5  6  7  8  9 10 11] [0 2]
[ 1  2  4  5  6  7  8  9 10 11] [0 3]
[ 1  2  3  5  6  7  8  9 10 11] [0 4]
[ 1  2  3  4  6  7  8  9 10 11] [0 5]

```


[1 2 3 4 5 7 8 9 10 11] [0 6]
[1 2 3 4 5 6 8 9 10 11] [0 7]
[1 2 3 4 5 6 7 9 10 11] [0 8]
[1 2 3 4 5 6 7 8 10 11] [0 9]
[1 2 3 4 5 6 7 8 9 11] [0 10]
[1 2 3 4 5 6 7 8 9 10] [0 11]
[0 3 4 5 6 7 8 9 10 11] [1 2]
[0 2 4 5 6 7 8 9 10 11] [1 3]
[0 2 3 5 6 7 8 9 10 11] [1 4]
[0 2 3 4 6 7 8 9 10 11] [1 5]
[0 2 3 4 5 7 8 9 10 11] [1 6]
[0 2 3 4 5 6 8 9 10 11] [1 7]
[0 2 3 4 5 6 7 9 10 11] [1 8]
[0 2 3 4 5 6 7 8 10 11] [1 9]
[0 2 3 4 5 6 7 8 9 11] [1 10]
[0 2 3 4 5 6 7 8 9 10] [1 11]
[0 1 4 5 6 7 8 9 10 11] [2 3]
[0 1 3 5 6 7 8 9 10 11] [2 4]
[0 1 3 4 6 7 8 9 10 11] [2 5]
[0 1 3 4 5 7 8 9 10 11] [2 6]
[0 1 3 4 5 6 8 9 10 11] [2 7]
[0 1 3 4 5 6 7 9 10 11] [2 8]
[0 1 3 4 5 6 7 8 10 11] [2 9]
[0 1 3 4 5 6 7 8 9 11] [2 10]
[0 1 3 4 5 6 7 8 9 10] [2 11]
[0 1 2 5 6 7 8 9 10 11] [3 4]
[0 1 2 4 6 7 8 9 10 11] [3 5]
[0 1 2 4 5 7 8 9 10 11] [3 6]
[0 1 2 4 5 6 8 9 10 11] [3 7]
[0 1 2 4 5 6 7 9 10 11] [3 8]
[0 1 2 4 5 6 7 8 10 11] [3 9]
[0 1 2 4 5 6 7 8 9 11] [3 10]
[0 1 2 4 5 6 7 8 9 10] [3 11]
[0 1 2 3 6 7 8 9 10 11] [4 5]
[0 1 2 3 5 7 8 9 10 11] [4 6]
[0 1 2 3 5 6 8 9 10 11] [4 7]
[0 1 2 3 5 6 7 9 10 11] [4 8]
[0 1 2 3 5 6 7 8 10 11] [4 9]
[0 1 2 3 5 6 7 8 9 11] [4 10]
[0 1 2 3 5 6 7 8 9 10] [4 11]
[0 1 2 3 4 7 8 9 10 11] [5 6]
[0 1 2 3 4 6 8 9 10 11] [5 7]
[0 1 2 3 4 6 7 9 10 11] [5 8]
[0 1 2 3 4 6 7 8 10 11] [5 9]
[0 1 2 3 4 6 7 8 9 11] [5 10]
[0 1 2 3 4 6 7 8 9 10] [5 11]
[0 1 2 3 4 5 8 9 10 11] [6 7]
[0 1 2 3 4 5 7 9 10 11] [6 8]
[0 1 2 3 4 5 7 8 10 11] [6 9]
[0 1 2 3 4 5 7 8 9 11] [6 10]

```

[ 0  1  2  3  4  5  7  8  9 10] [ 6 11]
[ 0  1  2  3  4  5  6  9 10 11] [7 8]
[ 0  1  2  3  4  5  6  8 10 11] [7 9]
[ 0  1  2  3  4  5  6  8  9 11] [ 7 10]
[ 0  1  2  3  4  5  6  8  9 10] [ 7 11]
[ 0  1  2  3  4  5  6  7 10 11] [8 9]
[ 0  1  2  3  4  5  6  7  9 11] [ 8 10]
[ 0  1  2  3  4  5  6  7  9 10] [ 8 11]
[ 0  1  2  3  4  5  6  7  8 11] [ 9 10]
[ 0  1  2  3  4  5  6  7  8 10] [ 9 11]
[0 1 2 3 4 5 6 7 8 9] [10 11]

```

```

X1 = range(12)
# Эквивалент KFold(n_splits=n)
kf = ShuffleSplit(n_splits=5, test_size=0.25)
for train, test in kf.split(X1):
    print("%s %s" % (train, test))

```

```

[ 3  6  4 10  7  1 11  8  9] [0 5 2]
[ 7  9  8  2 10  6  3  4  1] [ 0  5 11]
[ 0 11  3  6  7  2  5  8  1] [10  4  9]
[ 6 11  2 10  1  8  4  9  5] [0 3 7]
[ 2  8  5  6  0  7  9 10  3] [ 1  4 11]

```

```

X1 = np.ones(10)
y1 = [0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
skf = StratifiedKFold(n_splits=3)
for train, test in skf.split(X1, y1):
    print("%s %s" % (train, test))

```

```

[1 2 6 7 8 9] [0 3 4 5]
[0 2 3 4 5 8 9] [1 6 7]
[0 1 3 4 5 6 7] [2 8 9]

```

```

X1 = np.ones(10)
y1 = [0, 0, 0, 1, 1, 1, 1, 1, 1, 1]
skf = StratifiedKFold(n_splits=5)
for train, test in skf.split(X1, y1):
    print("%s %s" % (train, test))

```

```

[1 2 4 5 6 7 8 9] [0 3]
[0 2 3 5 6 7 8 9] [1 4]
[0 1 3 4 6 7 8 9] [2 5]
[0 1 2 3 4 5 8 9] [6 7]
[0 1 2 3 4 5 6 7] [8 9]

```

```

C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\model_selection\_split.py:676:
UserWarning: The least populated class in y has only 3 members, which
is less than n_splits=5.
    warnings.warn(

```

```

X1 = np.ones(10)
y1 = [0, 0, 0, 0, 1, 0, 1, 1, 1, 1]
skf = StratifiedKFold(n_splits=5)
for train, test in skf.split(X1, y1):
    print("%s %s" % (train, test))

[1 2 3 5 6 7 8 9] [0 4]
[0 2 3 4 5 7 8 9] [1 6]
[0 1 3 4 5 6 8 9] [2 7]
[0 1 2 4 5 6 7 9] [3 8]
[0 1 2 3 4 6 7 8] [5 9]

n_range = np.array(range(5,55,5))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters

[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}]

%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5,
scoring='accuracy')
clf_gs.fit(X_train, y_train)

Wall time: 1min 18s

GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 5, 10, 15, 20, 25,
30, 35, 40, 45, 50])}],
             scoring='accuracy')

clf_gs.cv_results_

{'mean_fit_time': array([0.21684804, 0.22814598, 0.22043943,
0.27976251, 0.25433254,
0.24078159, 0.24881349, 0.24733958, 0.22939444, 0.22700768]),
'std_fit_time': array([0.00161345, 0.00993246, 0.00173374,
0.04046872, 0.04037733,
0.0311306 , 0.03677876, 0.03107488, 0.0100987 , 0.00851473]),
'mean_score_time': array([1.03345842, 1.10205092, 1.14436364,
1.37666903, 1.31142654,
1.37678051, 1.44212422, 1.47185993, 1.55803823, 1.46546044]),
'std_score_time': array([0.0137137 , 0.0351537 , 0.04652715,
0.1496858 , 0.06285649,
0.1305565 , 0.11634956, 0.10507524, 0.07859415, 0.03810133]),
'param_n_neighbors': masked_array(data=[5, 10, 15, 20, 25, 30, 35,
40, 45, 50],
mask=[False, False, False, False, False, False, False,
False,
False, False],
fill_value='?',
dtype=object),
'params': [{'n_neighbors': 5},

```

```

{'n_neighbors': 10},
{'n_neighbors': 15},
{'n_neighbors': 20},
{'n_neighbors': 25},
{'n_neighbors': 30},
{'n_neighbors': 35},
{'n_neighbors': 40},
{'n_neighbors': 45},
{'n_neighbors': 50}],
'split0_test_score': array([0.75265229, 0.75914567, 0.76360113,
0.76501968, 0.76583884,
    0.76633833, 0.76685781, 0.76807656, 0.76809654, 0.76871591]),
'split1_test_score': array([0.75003496, 0.76038441, 0.76595872,
0.7671575 , 0.76883579,
    0.76919542, 0.76961499, 0.77155302, 0.77149308, 0.77195261]),
'split2_test_score': array([0.74947553, 0.75948532, 0.76495974,
0.76667799, 0.76875587,
    0.76883579, 0.77001459, 0.77143314, 0.7723522 , 0.77251204]),
'split3_test_score': array([0.74967033, 0.75858142, 0.7630969 ,
0.7645954 , 0.76781219,
    0.76597403, 0.76725275, 0.76801199, 0.76899101, 0.76899101]),
'split4_test_score': array([0.74455544, 0.75546454, 0.76045954,
0.76067932, 0.76341658,
    0.76535465, 0.76563437, 0.76595405, 0.76735265, 0.76637363]),
'mean_test_score': array([0.74927771, 0.75861227, 0.76361521,
0.76482598, 0.76693185,
    0.76713964, 0.7678749 , 0.76900575, 0.7696571 , 0.76970904]),
'std_test_score': array([0.00262505, 0.00167907, 0.00187227,
0.00228698, 0.00206295,
    0.00156781, 0.00167617, 0.00217001, 0.0019403 , 0.00225908]),
'rank_test_score': array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])}

```

Лучшая модель

clf_gs.best_estimator_

KNeighborsClassifier(n_neighbors=50)

Лучшее значение метрики

clf_gs.best_score_

0.7697090382395372

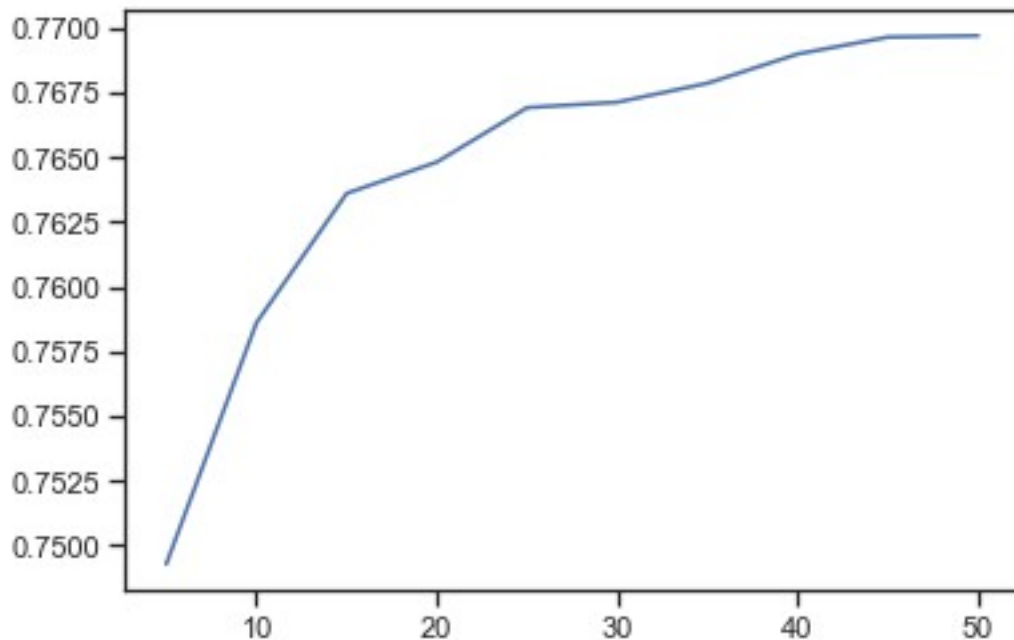
Лучшее значение параметров

clf_gs.best_params_

{'n_neighbors': 50}

Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])

[<matplotlib.lines.Line2D at 0x2931cb06760>]



```
%%time
clf_rs = RandomizedSearchCV(KNeighborsClassifier(), tuned_parameters,
cv=5, scoring='accuracy')
clf_rs.fit(X_train, y_train)

Wall time: 1min 17s

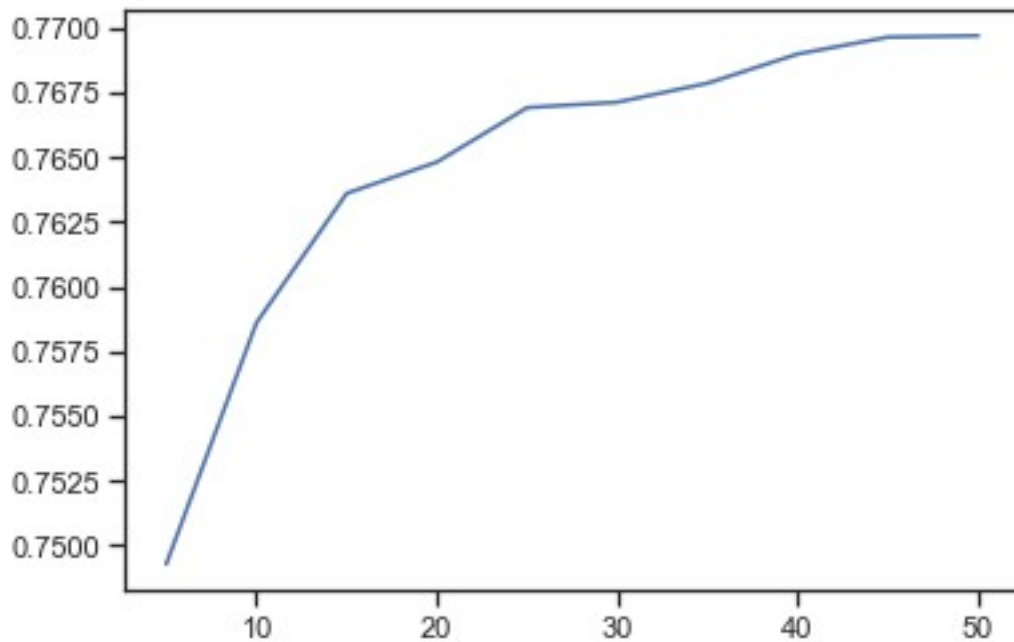
RandomizedSearchCV(cv=5, estimator=KNeighborsClassifier(),
                  param_distributions=[{'n_neighbors': array([ 5, 10,
15, 20, 25, 30, 35, 40, 45, 50])}],
                  scoring='accuracy')

# В данном случае оба способа нашли одинаковое решение
clf_rs.best_score_, clf_rs.best_params_
(0.7697090382395372, {'n_neighbors': 50})

clf_gs.best_score_, clf_gs.best_params_
(0.7697090382395372, {'n_neighbors': 50})

# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_rs.cv_results_['mean_test_score'])

[<matplotlib.lines.Line2D at 0x29325f6e700>]
```



```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0,
5)):
    """
    Generate a simple plot of the test and training learning curve.

    Parameters
    -----
    estimator : object type that implements the "fit" and "predict"
methods
        An object of that type which is cloned for each validation.

    title : string
        Title for the chart.

    X : array-like, shape (n_samples, n_features)
        Training vector, where n_samples is the number of samples and
        n_features is the number of features.

    y : array-like, shape (n_samples) or (n_samples, n_features),
optional
        Target relative to X for classification or regression;
        None for unsupervised learning.

    ylim : tuple, shape (ymin, ymax), optional
        Defines minimum and maximum yvalues plotted.

    cv : int, cross-validation generator or an iterable, optional
        Determines the cross-validation splitting strategy.
```

Possible inputs for cv are:

- None, to use the default 3-fold cross-validation,
- integer, to specify the number of folds.
- :term:`CV splitter`,
- An iterable yielding (train, test) splits as arrays of indices.

For integer/None inputs, if ``y`` is binary or multiclass, :class:`StratifiedKFold` used. If the estimator is not a classifier or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

Refer :ref:`User Guide <cross_validation>` for the various cross-validators that can be used here.

`n_jobs` : int or None, optional (default=None)
Number of jobs to run in parallel.
``None`` means 1 unless in a :obj:`joblib.parallel_backend` context.
``-1`` means using all processors. See :term:`Glossary <n_jobs>` for more details.

`train_sizes` : array-like, shape (n_ticks,), dtype float or int
Relative or absolute numbers of training examples that will be used to generate the learning curve. If the dtype is float, it is regarded as a fraction of the maximum size of the training set (that is determined by the selected validation method), i.e. it has to be within (0, 1]. Otherwise it is interpreted as absolute sizes of the training sets.
Note that for classification the number of samples usually have to be big enough to contain at least one sample from each class.
(default: `np.linspace(0.1, 1.0, 5)`)

```
"""
plt.figure()
plt.title(title)
if ylim is not None:
    plt.ylim(*ylim)
plt.xlabel("Training examples")
plt.ylabel("Score")
train_sizes, train_scores, test_scores = learning_curve(
    estimator, X, y, cv=cv, n_jobs=n_jobs,
    train_sizes=train_sizes)
train_scores_mean = np.mean(train_scores, axis=1)
```

```

train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)
plt.grid()

plt.fill_between(train_sizes, train_scores_mean -
train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.3,
                 color="r")
plt.fill_between(train_sizes, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.1,
color="g")
plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
         label="Training score")
plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
         label="Cross-validation score")

plt.legend(loc="best")
return plt

plot_learning_curve(KNeighborsClassifier(n_neighbors=5),
'n_neighbors=5',
                    X_train, y_train, cv=20)

```

```

C:\Users\Админ\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\LocalCache\local-
packages\Python39\site-packages\sklearn\model_selection\_split.py:676:
UserWarning: The least populated class in y has only 8 members, which
is less than n_splits=20.
  warnings.warn(

```

```

<module 'matplotlib.pyplot' from 'C:\\Users\\Админ\\AppData\\Local\\
Packages\\PythonSoftwareFoundation.Python.3.9_qbz5n2kfra8p0\\
LocalCache\\local-packages\\Python39\\site-packages\\matplotlib\\
pyplot.py'>

```