Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Дисциплина «Технологии машинного обучения»

Отчёт

по лабораторной работе №4

«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

Вариант 12

Студент:

Крюков Г. М.

Группа ИУ5-61Б

Преподаватель:

Гапанюк Ю. Е.

Москва, 2020 г.

Цель лабораторной работы:

Изучение сложных способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
3. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
4. Постройте модель и оцените качество модели с использованием кросс-валидации.
5. Произведите подбор гиперпараметра K с использованием GridSearchCV и кросс-валидации.

Выполнение работы:

Выбранный датасет:

Решение задачи классификации

`load_wine([return_X_y])` Load and return the wine dataset (classification).

https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_wine.html#sklearn.datasets.load_wine

1. Изучение качества классификации

```
[ ] import numpy as np
    import pandas as pd
    from typing import Dict, Tuple
    from scipy import stats
    from sklearn.datasets import load_wine, load_diabetes
    from sklearn.model_selection import train_test_split
    from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
    from sklearn.metrics import accuracy_score, balanced_accuracy_score
    from sklearn.metrics import plot_confusion_matrix
    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
    import seaborn as sns
    import matplotlib.pyplot as plt
    %matplotlib inline
    sns.set(style="ticks")
```

1.1. Подготовка данных и построение базовых моделей для оценки качества

```
[ ] wine = load_wine()
```

```
[ ] # Наименования признаков
    wine.feature_names
```

```
['alcohol',
 'malic_acid',
 'ash',
 'alcalinity_of_ash',
 'magnesium',
 'total_phenols',
 'flavanoids',
 'nonflavanoid_phenols',
 'proanthocyanins',
 'color_intensity',
 'hue',
 'od280/od315_of_diluted_wines',
 'proline']
```

```
[ ] # Значения признаков
    wine.data[:5]
```

```
array([[1.423e+01, 1.710e+00, 2.430e+00, 1.560e+01, 1.270e+02, 2.800e+00,
        3.060e+00, 2.800e-01, 2.290e+00, 5.640e+00, 1.040e+00, 3.920e+00,
        1.065e+03],
       [1.320e+01, 1.780e+00, 2.140e+00, 1.120e+01, 1.000e+02, 2.650e+00,
        2.760e+00, 2.600e-01, 1.280e+00, 4.380e+00, 1.050e+00, 3.400e+00,
        1.050e+03],
       [1.316e+01, 2.360e+00, 2.670e+00, 1.860e+01, 1.010e+02, 2.800e+00,
        3.240e+00, 3.000e-01, 2.810e+00, 5.680e+00, 1.030e+00, 3.170e+00,
        1.185e+03],
       [1.437e+01, 1.950e+00, 2.500e+00, 1.680e+01, 1.130e+02, 3.850e+00,
        3.490e+00, 2.400e-01, 2.180e+00, 7.800e+00, 8.600e-01, 3.450e+00,
        1.480e+03],
       [1.324e+01, 2.590e+00, 2.870e+00, 2.100e+01, 1.180e+02, 2.800e+00,
        2.690e+00, 3.900e-01, 1.820e+00, 4.320e+00, 1.040e+00, 2.930e+00,
        7.350e+02]])
```

```
[ ] type(wine.data)
```

```
numpy.ndarray
```

```
[ ] # Значения целевого признака
    np.unique(wine.target)
```

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

```
[ ] # Наименования значений целевого признака
    wine.target_names
```

```
array(['class_0', 'class_1', 'class_2'], dtype='<U7')
```

```
[16] list(zip(np.unique(wine.target), wine.target_names))
```

```
[(0, 'class_0'), (1, 'class_1'), (2, 'class_2')]
```

```
[17] # Значения целевого признака
     wine.target
```

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

```
[18] # Размер выборки
     wine.data.shape, wine.target.shape
```

```
((178, 13), (178,))
```

Формируем DataFrame:

```
[19] # Сформируем DataFrame
     wine_df = pd.DataFrame(data= np.c_[wine['data'], wine['target']],
                            columns= wine['feature_names'] + ['target'])
```

```
# И выведем его статистические характеристики
wine_df.describe()
```

```
[19] # Сформируем DataFrame
     wine_df = pd.DataFrame(data= np.c_[wine['data'], wine['target']],
                            columns= wine['feature_names'] + ['target'])
```

```
# И выведем его статистические характеристики
wine_df.describe()
```

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od28
count	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	178.000000	
mean	13.000618	2.336348	2.366517	19.494944	99.741573	2.295112	2.029270	0.361854	1.590899	5.058090	0.957449	
std	0.811827	1.117146	0.274344	3.339564	14.282484	0.625851	0.998859	0.124453	0.572359	2.318286	0.228572	
min	11.030000	0.740000	1.360000	10.600000	70.000000	0.980000	0.340000	0.130000	0.410000	1.280000	0.480000	
25%	12.362500	1.602500	2.210000	17.200000	88.000000	1.742500	1.205000	0.270000	1.250000	3.220000	0.782500	
50%	13.050000	1.865000	2.360000	19.500000	98.000000	2.355000	2.135000	0.340000	1.555000	4.690000	0.965000	
75%	13.677500	3.082500	2.557500	21.500000	107.000000	2.800000	2.875000	0.437500	1.950000	6.200000	1.120000	
max	14.830000	5.800000	3.230000	30.000000	162.000000	3.880000	5.080000	0.660000	3.580000	13.000000	1.710000	

1.2. Разделение выборки на обучающую и тестовую

```
[21] wine_X_train, wine_X_test, wine_y_train, wine_y_test = train_test_split
         wine.data, wine.target, test_size=0.5, random_state=1)
```

```
[22] # Размер обучающей выборки
     wine_X_train.shape, wine_y_train.shape
```

```
((89, 13), (89,))
```

```
[23] # Размер тестовой выборки
     wine_X_test.shape, wine_y_test.shape
```

```
((89, 13), (89,))
```

```
[24] np.unique(wine_y_train)
```

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

```
np.unique(wine_y_test)
```

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

1.3. Построим базовые модели на основе метода ближайших соседей

```
[27] # 2 ближайших соседа
     cl1_1 = KNeighborsClassifier(n_neighbors=2)
     cl1_1.fit(wine_X_train, wine_y_train)
     target1_1 = cl1_1.predict(wine_X_test)
     len(target1_1), target1_1
```

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

```
[31] # 3 ближайших соседа
     cl1_1 = KNeighborsClassifier(n_neighbors=3)
     cl1_1.fit(wine_X_train, wine_y_train)
     target1_1 = cl1_1.predict(wine_X_test)
     len(target1_1), target1_1
```

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

```
# 10 ближайших соседей
cl1_2 = KNeighborsClassifier(n_neighbors=10)
cl1_2.fit(wine_X_train, wine_y_train)
target1_2 = cl1_2.predict(wine_X_test)
len(target1_2), target1_2
```

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

2. Метрики качества классификации

2.1. Accuracy

```
[34] # 3 ближайших соседа
     accuracy_score(wine_y_test, target1_1)
```

```
0.6966292134831461
```

```
# 10 ближайших соседей
accuracy_score(wine_y_test, target1_2)
```

```
0.6629213483146067
```

Точность в случае 10 ближайших соседей составляет более 66%, а точность в случае 2 ближайших соседей составляет более 69%.

```python
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:

    # 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_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res
```

```python
def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy 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]))
```

[46]
```python
# 3 ближайших соседа
print_accuracy_score_for_classes(wine_y_test, target1_1)
```

```
Метка    Accuracy
0        0.7878787878787878
1        0.7352941176470589
2        0.5
```

```python
# 10 ближайших соседей
print_accuracy_score_for_classes(wine_y_test, target1_2)
```

```
Метка    Accuracy
0        0.7878787878787878
1        0.8235294117647058
2        0.22727272727272727
```

```python
# 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_wine_y_test = convert_target_to_binary(wine_y_test, 2)

# Конвертация предсказанных признаков
bin_target1_1 = convert_target_to_binary(target1_1, 2)
bin_target1_2 = convert_target_to_binary(target1_2, 2)
```

[51]
```python
balanced_accuracy_score(bin_wine_y_test, bin_target1_1)
```

```
0.6529850746268657
```

```python
balanced_accuracy_score(bin_wine_y_test, bin_target1_2)
```

```
0.5390094979647219
```

2.2. Матрица ошибок или ConfusionMatrix

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

```
array([[54, 13],
       [11, 11]])
```

```
[55] tn, fp, fn, tp = confusion_matrix(bin_wine_y_test, bin_target1_1).ravel()
     tn, fp, fn, tp
```

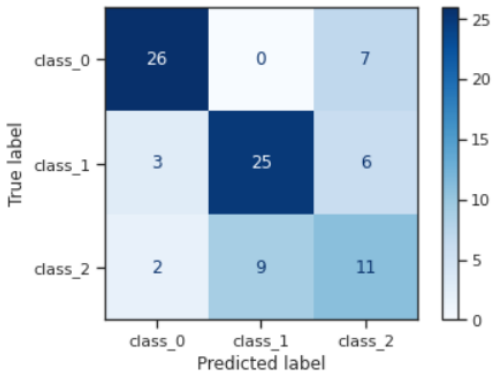
```
(54, 13, 11, 11)
```

```
[57] # Пример для небинарной классификации
     confusion_matrix(wine_y_test, target1_1, labels=[0, 1, 2])
```

```
array([[26,  0,  7],
       [ 3, 25,  6],
       [ 2,  9, 11]])
```

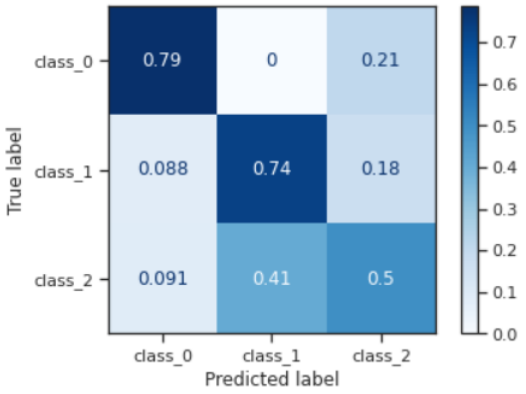
```
[58] plot_confusion_matrix(cl1_1, wine_X_test, wine_y_test,
                           display_labels=wine.target_names, cmap=plt.cm.Blues)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f45b4cd9cc0>
```




```
[59] plot_confusion_matrix(cl1_1, wine_X_test, wine_y_test,
                           display_labels=wine.target_names,
                           cmap=plt.cm.Blues, normalize='true')
```

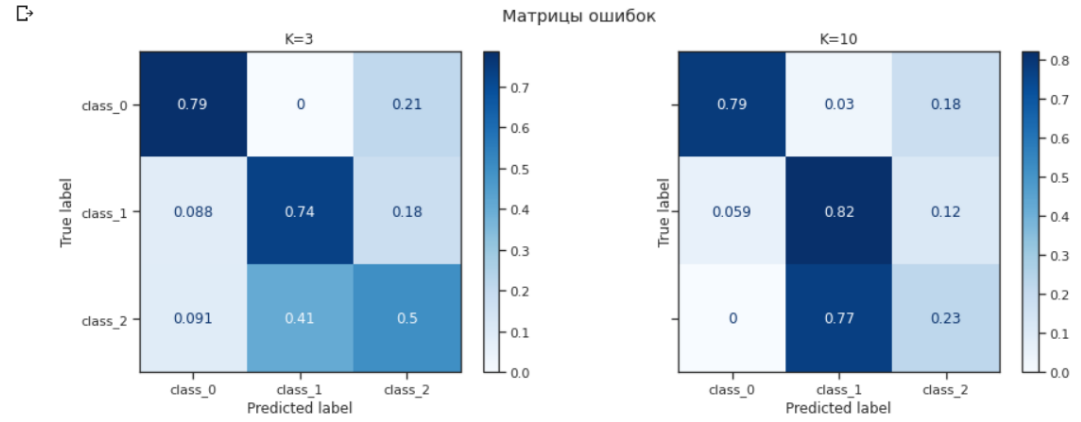
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f45b4bdf320>
```



```
plot_confusion_matrix(cl1_1, wine_X_test, wine_y_test,
                      display_labels=wine.target_names,
                      cmap=plt.cm.Blues, normalize='true', ax=ax[0])

plot_confusion_matrix(cl1_2, wine_X_test, wine_y_test,
                      display_labels=wine.target_names,
                      cmap=plt.cm.Blues, normalize='true', ax=ax[1])

fig.suptitle('Матрицы ошибок')
ax[0].title.set_text('K=3')
ax[1].title.set_text('K=10')
```



2.3. Precision, recall и F-мера

```
[61] # По умолчанию метрики считаются для 1 класса бинарной классификации
     # Для 3 ближайших соседей
     precision_score(bin_wine_y_test, bin_target1_1), recall_score(bin_wine_y_test, bin_target1_1)
```

```
(0.4583333333333333, 0.5)
```

```
[62] # Для 10 ближайших соседей
     precision_score(bin_wine_y_test, bin_target1_2), recall_score(bin_wine_y_test, bin_target1_2)
```

```
(0.3333333333333333, 0.22727272727272727)
```

```
[63] # Параметры TP, TN, FP, FN считаются как сумма по всем классам
     precision_score(wine_y_test, target1_1, average='micro')
```

```
0.6966292134831461
```

```
[64] # Параметры TP, TN, FP, FN считаются отдельно для каждого класса
     # и берется среднее значение, дисбаланс классов не учитывается.
     precision_score(wine_y_test, target1_1, average='macro')
```

```
0.6774457094665824
```

```
# 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(wine_y_test, target1_1, average='weighted')
```

```
0.7051769964963152
```

```
[66] f1_score(bin_wine_y_test, bin_target1_2)
```

```
0.27027027027027023
```

```
[67] f1_score(wine_y_test, target1_1, average='micro')
```

```
0.6966292134831461
```

```
[68] f1_score(wine_y_test, target1_1, average='macro')
```

```
0.6753516624040921
```

```
[69] f1_score(wine_y_test, target1_1, average='weighted')
```

```
0.7003847093307279
```

```
classification_report(wine_y_test, target1_1,
                      target_names=wine.target_names, output_dict=True)
```

```
{'accuracy': 0.6966292134831461,
 'class_0': {'f1-score': 0.8125,
  'precision': 0.8387096774193549,
  'recall': 0.7878787878787878,
  'support': 33},
 'class_1': {'f1-score': 0.735294117647059,
  'precision': 0.7352941176470589,
  'recall': 0.7352941176470589,
  'support': 34},
 'class_2': {'f1-score': 0.4782608695652174,
  'precision': 0.4583333333333333,
  'recall': 0.5,
  'support': 22},
 'macro avg': {'f1-score': 0.6753516624040921,
  'precision': 0.6774457094665824,
  'recall': 0.6743909685086157,
  'support': 89},
 'weighted avg': {'f1-score': 0.7003847093307279,
  'precision': 0.7051769964963152,
  'recall': 0.6966292134831461,
  'support': 89}}
```

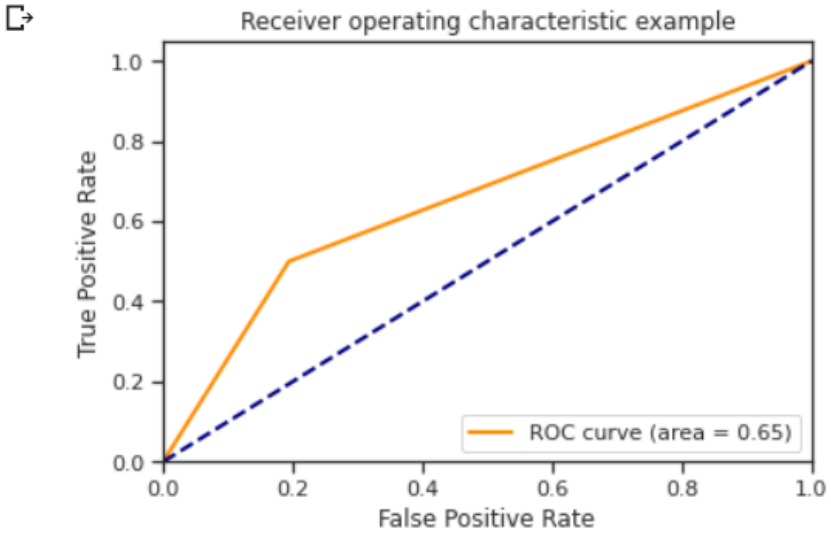
2.4. ROC-кривая и ROC AUC

```
[71] fpr, tpr, thresholds = roc_curve(bin_wine_y_test, bin_target1_1,
                                      pos_label=1)
     fpr, tpr, thresholds
```

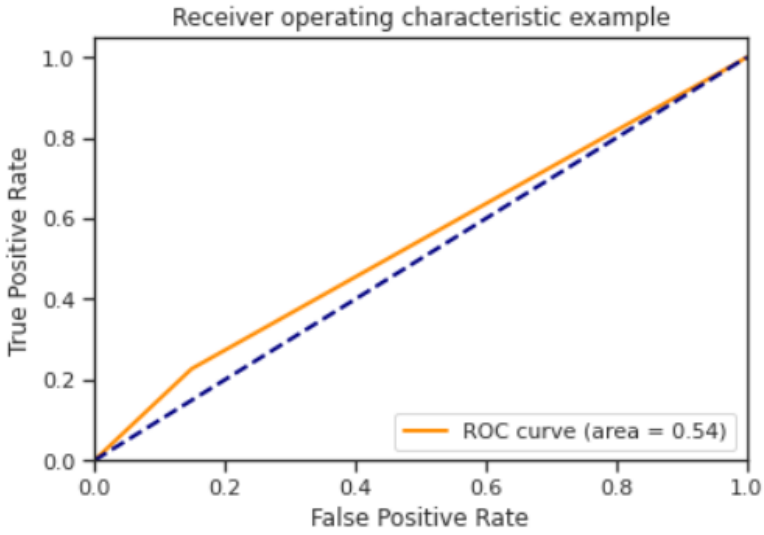
```
(array([0.        , 0.19402985, 1.        ]),
 array([0. , 0.5, 1. ]),
 array([2, 1, 0]))
```

```
[72] # 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 3 ближайших соседей
draw_roc_curve(bin_wine_y_test, bin_target1_1, pos_label=1, average='micro')
```



```
# Для 10 ближайших соседей
draw_roc_curve(bin_wine_y_test, bin_target1_2, pos_label=1, average='micro')
```



3. Разбиение выборки на k частей с помощью кросс-валидации.

Наиболее простым способом кросс-валидации является вызов функции cross_val_score. В этом случае стратегия кросс-валидации определяется автоматически.

```
[78] from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
     from sklearn.model_selection import cross_val_score, cross_validate
```

```
[80] scores = cross_val_score(KNeighborsClassifier(n_neighbors=3),
                              wine.data, wine.target, cv=3)
```

```
[81] # Значение метрики accuracy для 3 фолдов
     scores
```

```
array([0.61666667, 0.57627119, 0.79661017])
```

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

```
0.6631826741996233
```

В отличие от функции cross_val_score, функция cross_validate позволяет использовать для оценки несколько метрик и возращает более детальную информацию.

```
[83] scoring = {'precision': 'precision_weighted',
                'recall': 'recall_weighted',
                'f1': 'f1_weighted'}
```

```
scores = cross_validate(KNeighborsClassifier(n_neighbors=3),
                        wine.data, wine.target, scoring=scoring,
                        cv=3, return_train_score=True)
scores
```

```
{'fit_time': array([0.00085354, 0.00051498, 0.00043821]),
 'score_time': array([0.00853586, 0.00439715, 0.00407171]),
 'test_f1': array([0.60380952, 0.57731661, 0.78515946]),
 'test_precision': array([0.61631579, 0.59473992, 0.78926159]),
 'test_recall': array([0.61666667, 0.57627119, 0.79661017]),
 'train_f1': array([0.83420614, 0.87517928, 0.78484177]),
 'train_precision': array([0.84432192, 0.87868395, 0.78719633]),
 'train_recall': array([0.83050847, 0.87394958, 0.78991597])}
```

4. Нахождение наилучшего гиперпараметра K с использованием GridSearchCV и кросс-валидации

```
n_range = np.array(range(5,30,1))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21,
         22, 23, 24, 25, 26, 27, 28, 29])}]
```

```
[91] from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
```

```
[94] %%time
     clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accuracy')
     clf_gs.fit(wine_X_train, wine_y_train)
```

```
CPU times: user 262 ms, sys: 0 ns, total: 262 ms
Wall time: 265 ms
```

```
clf_gs.cv_results_
```

```
{'mean_fit_time': array([0.00079412, 0.00042229, 0.000424  , 0.00041122, 0.00042453,
       0.00041642, 0.00042357, 0.00042272, 0.00042644, 0.00042472,
       0.00041595, 0.00041776, 0.00041423, 0.00041547, 0.00041981,
       0.0004138 , 0.00041986, 0.00041375, 0.00042772, 0.00042868,
       0.00042977, 0.00041394, 0.00041647, 0.00042043, 0.00040278]),
 'mean_score_time': array([0.00209603, 0.00128002, 0.00122681, 0.00126824, 0.00124025,
       0.00131435, 0.0012342 , 0.00125484, 0.00124636, 0.00125728,
       0.0012476 , 0.00128284, 0.00126023, 0.00125666, 0.00125179,
       0.00125437, 0.00125766, 0.0013196 , 0.00127416, 0.00133877,
       0.00129719, 0.00133085, 0.00127959, 0.00134783, 0.00130792]),
 'mean_test_score': array([0.69411765, 0.67254902, 0.63856209, 0.62679739, 0.66143791,
       0.68300654, 0.70718954, 0.69607843, 0.69607843, 0.70784314,
       0.71895425, 0.71830065, 0.74052288, 0.71830065, 0.71830065,
       0.70784314, 0.69607843, 0.69607843, 0.68431373, 0.69607843,
       0.71699346, 0.73006536, 0.68366013, 0.63921569, 0.69477124]),
 'param_n_neighbors': masked_array(data=[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
                    20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
              mask=[False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False, False, False, False, False, False, False, False,
                    False],
        fill_value='?',
             dtype=object),
 'params': [{'n_neighbors': 5},
  {'n_neighbors': 6},
  {'n_neighbors': 7},
  {'n_neighbors': 8},
  {'n_neighbors': 9},
```

```
clf_gs.cv_results_
```

```
              dtype=object),
 'params': [{'n_neighbors': 5},
  {'n_neighbors': 6},
  {'n_neighbors': 7},
  {'n_neighbors': 8},
  {'n_neighbors': 9},
  {'n_neighbors': 10},
  {'n_neighbors': 11},
  {'n_neighbors': 12},
  {'n_neighbors': 13},
  {'n_neighbors': 14},
  {'n_neighbors': 15},
  {'n_neighbors': 16},
  {'n_neighbors': 17},
  {'n_neighbors': 18},
  {'n_neighbors': 19},
  {'n_neighbors': 20},
  {'n_neighbors': 21},
  {'n_neighbors': 22},
  {'n_neighbors': 23},
  {'n_neighbors': 24},
  {'n_neighbors': 25},
  {'n_neighbors': 26},
  {'n_neighbors': 27},
  {'n_neighbors': 28},
  {'n_neighbors': 29}],
 'rank_test_score': array([17, 21, 24, 25, 22, 20, 10, 11, 11,  9,  3,  4,  1,  4,  6,  8, 11,
        11, 18, 11,  7,  2, 19, 23, 16], dtype=int32),
 'split0_test_score': array([0.77777778, 0.66666667, 0.77777778, 0.72222222, 0.77777778,
```

```
 {'n_neighbors': 29}],
'rank_test_score': array([17, 21, 24, 25, 22, 20, 10, 11, 11,  9,  3,  4,  1,  4,  6,  8, 11,
       11, 18, 11,  7,  2, 19, 23, 16], dtype=int32),
'split0_test_score': array([0.77777778, 0.66666667, 0.77777778, 0.72222222, 0.77777778,
       0.77777778, 0.72222222, 0.77777778, 0.72222222, 0.72222222,
       0.83333333, 0.77777778, 0.77777778, 0.72222222, 0.72222222,
       0.72222222, 0.72222222, 0.72222222, 0.72222222, 0.66666667,
       0.77777778, 0.72222222, 0.66666667, 0.61111111, 0.72222222]),
'split1_test_score': array([0.72222222, 0.72222222, 0.61111111, 0.66666667, 0.61111111,
       0.66666667, 0.66666667, 0.66666667, 0.61111111, 0.72222222,
       0.72222222, 0.77777778, 0.72222222, 0.77777778, 0.72222222,
       0.66666667, 0.66666667, 0.61111111, 0.61111111, 0.61111111,
       0.77777778, 0.77777778, 0.77777778, 0.77777778, 0.77777778]),
'split2_test_score': array([0.66666667, 0.66666667, 0.61111111, 0.61111111, 0.61111111,
       0.72222222, 0.72222222, 0.66666667, 0.66666667, 0.72222222,
       0.61111111, 0.66666667, 0.77777778, 0.72222222, 0.72222222,
       0.72222222, 0.66666667, 0.72222222, 0.72222222, 0.83333333,
       0.72222222, 0.66666667, 0.72222222, 0.55555556, 0.72222222]),
'split3_test_score': array([0.83333333, 0.77777778, 0.72222222, 0.72222222, 0.77777778,
       0.77777778, 0.77777778, 0.72222222, 0.83333333, 0.66666667,
       0.72222222, 0.72222222, 0.77777778, 0.72222222, 0.77777778,
       0.72222222, 0.77777778, 0.77777778, 0.77777778, 0.72222222,
       0.77777778, 0.77777778, 0.72222222, 0.72222222, 0.72222222]),
'split4_test_score': array([0.47058824, 0.52941176, 0.47058824, 0.41176471, 0.52941176,
       0.47058824, 0.64705882, 0.64705882, 0.64705882, 0.70588235,
       0.70588235, 0.64705882, 0.64705882, 0.64705882, 0.64705882,
       0.70588235, 0.64705882, 0.64705882, 0.58823529, 0.64705882,
       0.52941176, 0.70588235, 0.52941176, 0.52941176, 0.52941176]),
'std_fit_time': array([2.95198766e-04, 9.22824666e-06, 1.16397113e-05, 3.10048306e-05,
       9.08571883e-06, 6.74282138e-06, 6.14657709e-06, 3.13771894e-06,
       1.34773786e-05, 2.23313588e-05, 5.49625932e-06, 6.36787416e-06,
```



```
       0.77777778, 0.77777778, 0.72222222, 0.72222222, 0.72222222]),
 'split4_test_score': array([0.47058824, 0.52941176, 0.47058824, 0.41176471, 0.52941176,
       0.47058824, 0.64705882, 0.64705882, 0.64705882, 0.70588235,
       0.70588235, 0.64705882, 0.64705882, 0.64705882, 0.64705882,
       0.70588235, 0.64705882, 0.64705882, 0.58823529, 0.64705882,
       0.52941176, 0.70588235, 0.52941176, 0.52941176, 0.52941176]),
 'std_fit_time': array([2.95198766e-04, 9.22824666e-06, 1.16397113e-05, 3.10048306e-05,
       9.08571883e-06, 6.74282138e-06, 6.14657709e-06, 3.13771894e-06,
       1.34773786e-05, 2.23313588e-05, 5.49625932e-06, 6.36787416e-06,
       3.25019706e-06, 6.83825051e-06, 6.73573628e-06, 6.64088835e-06,
       9.81636204e-06, 5.37071918e-06, 4.93013650e-06, 7.43774262e-06,
       6.46215515e-06, 9.57291419e-06, 7.43835400e-06, 1.92278339e-06,
       6.96671980e-06]),
 'std_score_time': array([5.87245348e-04, 3.78653142e-05, 3.09571996e-05, 6.63457863e-05,
       3.33649744e-05, 1.34349104e-04, 8.77769011e-06, 2.59980784e-05,
       5.56205558e-06, 3.01392016e-05, 2.22910024e-05, 9.26794812e-05,
       1.87594065e-05, 1.62610560e-05, 1.40190125e-05, 1.78482384e-05,
       1.93897492e-05, 9.92235455e-05, 1.47514644e-05, 4.50280235e-05,
       2.36187991e-05, 3.74661730e-05, 1.45962027e-05, 7.98712061e-05,
       3.03428835e-05]),
 'std_test_score': array([0.12481093, 0.08258087, 0.10592672, 0.11514032, 0.09956546,
       0.11392065, 0.04624385, 0.04794019, 0.07747235, 0.0215389 ,
       0.07057613, 0.05446449, 0.0514475 , 0.04161505, 0.04161505,
       0.0215389 , 0.04794019, 0.05943761, 0.07238691, 0.07747235,
       0.09622726, 0.04293874, 0.08475086, 0.09584918, 0.08543361])}
```

```
[96] # Лучшая модель
     clf_gs.best_estimator_
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=17, p=2,
                     weights='uniform')
```

```
[97] # Лучшее значение метрики
     clf_gs.best_score_
```

```
0.7405228758169934
```

```
# Лучшее значение параметров
clf_gs.best_params_
```

```
{'n_neighbors': 17}
```

Лучшее значение гиперпараметра =17

Лучшее значение метрики при этом гиперпараметре = 0.7405228758169934

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

```
[<matplotlib.lines.Line2D at 0x7f45b43afe80>]
```