Задание: Выберите набор данных (датасет) для решения задачи классификации или регрессии. С использованием метода train_test_split разделите выборку на обучающую и тестовую. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик. Произведите подбор гиперпараметра K с использованием GridSearchCV и/или RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Желательно использование нескольких стратегий кросс-валидации. Сравните метрики качества исходной и оптимальной моделей.

In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import learning_curve, validation_curve
import sklearn.metrics as mtr
import matplotlib.pyplot as plt
```

In [ ]:

```
wine = load_wine()

wine.feature_names
```

Out[ ]:

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

In [ ]:

```
np.unique(wine.target)
```

Out[ ]:

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

In [ ]:

```
wine.target
```

Out[ ]:

```
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, 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])
```

In [ ]:

```
wine.target_names
```

Out[ ]:

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

In [ ]:

```
wine.data.shape, wine.target.shape
```

Out[ ]:

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

In [ ]:

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

In [ ]:

```
wine_df.describe()
```

Out[ ]:

|  | alcohol | malic_acid | ash | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity |
|---|---|---|---|---|---|---|---|---|---|---|
| **count** | 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 |
| **std** | 0.811827 | 1.117146 | 0.274344 | 3.339564 | 14.282484 | 0.625851 | 0.998859 | 0.124453 | 0.572359 | 2.318286 |
| **min** | 11.030000 | 0.740000 | 1.360000 | 10.600000 | 70.000000 | 0.980000 | 0.340000 | 0.130000 | 0.410000 | 1.280000 |
| **25%** | 12.362500 | 1.602500 | 2.210000 | 17.200000 | 88.000000 | 1.742500 | 1.205000 | 0.270000 | 1.250000 | 3.220000 |
| **50%** | 13.050000 | 1.865000 | 2.360000 | 19.500000 | 98.000000 | 2.355000 | 2.135000 | 0.340000 | 1.555000 | 4.690000 |
| **75%** | 13.677500 | 3.082500 | 2.557500 | 21.500000 | 107.000000 | 2.800000 | 2.875000 | 0.437500 | 1.950000 | 6.200000 |
| **max** | 14.830000 | 5.800000 | 3.230000 | 30.000000 | 162.000000 | 3.880000 | 5.080000 | 0.660000 | 3.580000 | 13.000000 |

In [ ]:

```
# Формирование обучающей и тестовой выборки
wine_X_train, wine_X_test, wine_Y_train, wine_Y_test = train_test_split(wine.data, wine.target, test_size=0.2, random_state=1)
wine_X_train.shape, wine_X_test.shape, wine_Y_train.shape, wine_Y_test.shape
```

Out[ ]:

```
((142, 13), (36, 13), (142,), (36,))
```

In [ ]:

```
# Первичное обучение модели и оценка качества
cls_simple = KNeighborsClassifier(n_neighbors=5)
cls_simple.fit(wine_X_train, wine_Y_train)
target_simple = cls_simple.predict(wine_X_test)
target_simple_train_prediction = cls_simple.predict(wine_X_train)
target_simple
```

Out[ ]:

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

In [ ]:

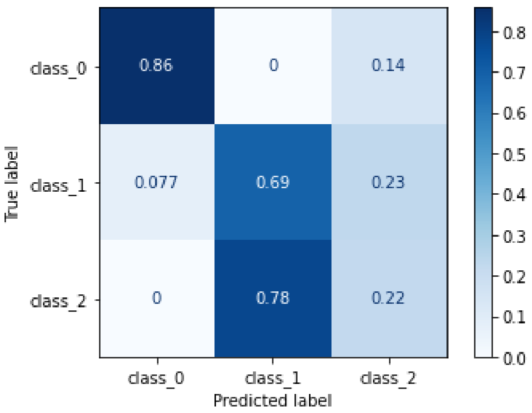
```
mtr.accuracy_score(wine_Y_test, target_simple)
```

Out[ ]:

```
0.6388888888888888
```

In [ ]:

```
matrix = mtr.ConfusionMatrixDisplay.from_predictions(wine_Y_test, target_simple, display_labels=wine.target_names, cmap=plt.cm.Blues, normalize='true')
```



In [ ]:

```
mtr.f1_score(wine_Y_test, target_simple, average='micro')
```

Out[ ]:

```
0.6388888888888888
```

In [ ]:

```
# Оценка качества модели с использованием кросс-валидации
scores = cross_val_score(KNeighborsClassifier(n_neighbors=5),
                         wine.data, wine.target,
                         cv=LeaveOneOut())
scores, np.mean(scores)
```

Out[ ]:

```
(array([1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1.,
       0., 0., 0., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 0., 0.,
       1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 0., 0., 1., 0., 1., 0., 0.,
       1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1.,
       0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0.,
       1., 1., 0., 1., 1., 0., 1., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0.,
       0., 1., 1., 0., 0., 1., 0., 1., 0., 0., 0., 1., 1., 1., 0., 0., 0.,
       1., 0., 1., 1., 0., 1., 1., 0.]), 0.6966292134831461)
```

In [ ]:

```
# Подбор гиперпараметров на основе решетчатого поиска и кросс-валидации

n_range = np.array(range(1, 50, 4))
tuned_parameters = [{"n_neighbors": n_range}]

grid_search = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=LeaveOneOut(), scoring='accuracy')
grid_search.fit(wine.data, wine.target)

grid_search.best_params_
```

Out[ ]:

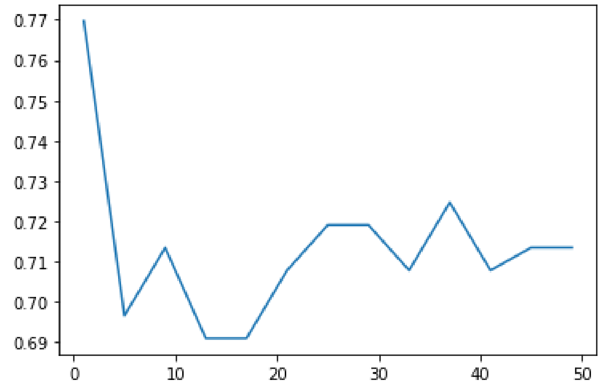
```
{'n_neighbors': 1}
```

In [ ]:

```
plt.plot(n_range, grid_search.cv_results_['mean_test_score'])
```

Out[ ]:

```
[<matplotlib.lines.Line2D at 0x7fb77b8d3210>]
```



In [ ]:

```
# Обучение модели и оценка качества с учетом подобранных гиперпараметров
grid_search.best_estimator_.fit(wine_X_train, wine_Y_train)
target2_0 = grid_search.best_estimator_.predict(wine_X_train)
target2_1 = grid_search.best_estimator_.predict(wine_X_test)

# Новое качество модели
mtr.accuracy_score(wine_Y_train, target2_0), mtr.accuracy_score(wine_Y_test, target2_1)
```

Out[ ]:

```
(1.0, 0.75)
```

In [ ]:

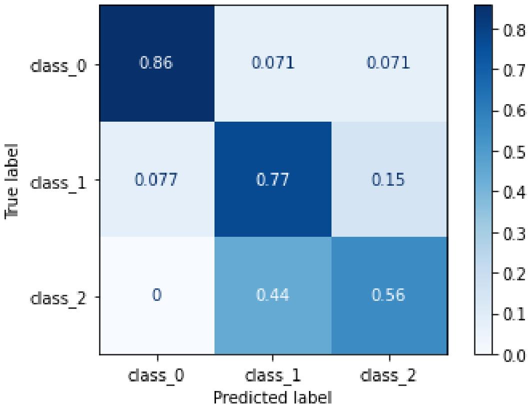
```
# Качество модели до подбора гиперпараметров
mtr.accuracy_score(wine_Y_train, target_simple_train_prediction), mtr.accuracy_score(wine_Y_test, target_simple)
```

Out[ ]:

```
(0.823943661971831, 0.6388888888888888)
```

In [ ]:

```
# Новое качество модели
matrix = mtr.ConfusionMatrixDisplay.from_predictions(wine_Y_test, target2_1, display_labels=wine.target_names, cmap=plt.cm.Blues, normalize='true')
```



In [ ]:

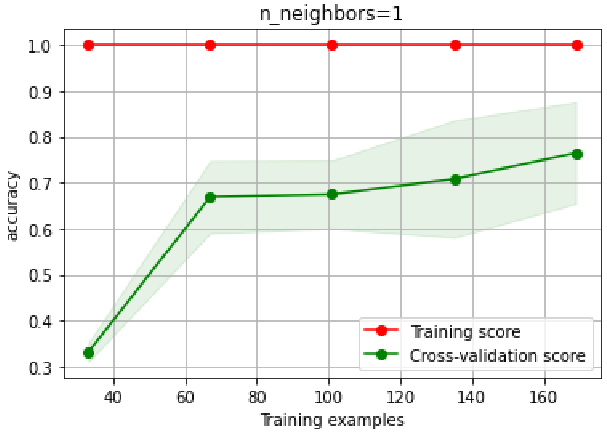
```
# 6.Построение кривых обучения и валидации

def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5), scoring='accuracy'):
    """
    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(scoring)
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=cv, scoring=scoring, 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
```

In [ ]:

```
plot_learning_curve(grid_search.best_estimator_, 'n_neighbors=1',
                    wine.data, wine.target, cv=20, train_sizes=np.linspace(.2, 1.0, 5))
```

Out[ ]:

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.7/dist-packages/matplotlib/pyplot.py'>
```



In [ ]:

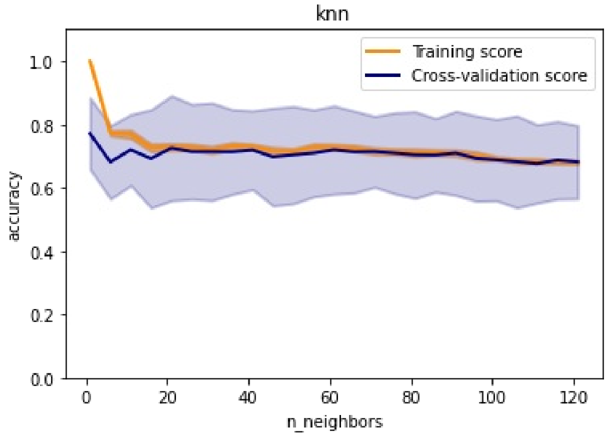
```
def plot_validation_curve(estimator, title, X, y,
                          param_name, param_range, cv,
                          scoring='accuracy'):

    train_scores, test_scores = validation_curve(
        estimator, X, y, param_name=param_name, param_range=param_range,
        cv=cv, scoring=scoring, n_jobs=1)
    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.title(title)
    plt.xlabel(param_name)
    plt.ylabel(str(scoring))
    plt.ylim(0.0, 1.1)
    lw = 2
    plt.plot(param_range, train_scores_mean, label="Training score",
                 color="darkorange", lw=lw)
    plt.fill_between(param_range, train_scores_mean - train_scores_std,
                     train_scores_mean + train_scores_std, alpha=0.4,
                     color="darkorange", lw=lw)
    plt.plot(param_range, test_scores_mean, label="Cross-validation score",
                 color="navy", lw=lw)
    plt.fill_between(param_range, test_scores_mean - test_scores_std,
                     test_scores_mean + test_scores_std, alpha=0.2,
                     color="navy", lw=lw)
    plt.legend(loc="best")
    return plt
```

In [ ]:

```
n_range2 = np.array(range(1,125,5))
plot_validation_curve(grid_search.best_estimator_, 'knn',
                      wine.data, wine.target,
                      param_name='n_neighbors', param_range=n_range2,
                      cv=20, scoring="accuracy")
```

Out[ ]:

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.7/dist-packages/matplotlib/pyplot.py'>
```


knn
Training score
Cross-validation score
accuracy
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100
120
n_neighbors