Московский государственный технический

университет им. Н. Э. Баумана

Факультет «Радиотехнический»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения»

Отчет по лабораторной работе №3
«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

Выполнил:
студент группы РТ5-61Б

Алиев Тимур

Подпись и дата:

Проверил:
доцент каф. ИУ5

Гапанюк Ю.Е.

Подпись и дата:

Москва, 2022 г.

Описание задания

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. Произведите подбор гиперпараметра K с использованием GridSearchCV и/или RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Желательно использование нескольких стратегий кросс-валидации.
5. Сравните метрики качества исходной и оптимальной моделей.

Jupyter notebook

Подготовка данных

Ввод [1]:
```
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
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
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")
```

Ввод [2]:
```
# https://scikit-learn.org/stable/datasets/index.html#iris-dataset
iris = load_iris()
```

Ввод [3]:
```
# Наименования признаков
iris.feature_names
```

Out[3]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

Ввод [4]:
```
# Значения признаков
iris.data[:5]
```

Out[4]:
```
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
```

Ввод [5]:
```
type(iris.data)
```
Out[5]:
```
numpy.ndarray
```

Ввод [6]:
```
# Значения целевого признака
np.unique(iris.target)
```
Out[6]:
```
array([0, 1, 2])
```

Ввод [7]:
```
# Наименования значений целевого признака
iris.target_names
```
Out[7]:
```
array(['setosa', 'versicolor', 'virginica'], dtype='<U10')
```

Ввод [8]:
```
list(zip(np.unique(iris.target), iris.target_names))
```
Out[8]:
```
[(0, 'setosa'), (1, 'versicolor'), (2, 'virginica')]
```

Ввод [9]:
```
# Значения целевого признака
iris.target
```
Out[9]:
```
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, 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, 2, 2])
```

Ввод [10]:
```
# Размер выборки
iris.data.shape, iris.target.shape
```
Out[10]:
```
((150, 4), (150,))
```

Ввод [11]:
```
# Сформируем DataFrame
iris_df = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                       columns= iris['feature_names'] + ['target'])
```

Ввод [12]:
```
# И выведем его статистические характеристики
iris_df.describe()
```
Out[12]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333	1.000000
std	0.828066	0.435866	1.765298	0.762238	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

Разделение выборки на обучающую и тестовую

Ввод [13]:
```
# Как правило, параметр test_size устанавливают в 20% или 30%.
# Здесь используется test_size=0.5 (50%) чтобы искусственно "ухудшить"
# результат на хорошем наборе данных и показать различные возможности
# использования метрик качества.
iris_X_train, iris_X_test, iris_y_train, iris_y_test = train_test_split(
    iris.data, iris.target, test_size=0.5, random_state=1)
```

Ввод [14]:
```
# Размер обучающей выборки
iris_X_train.shape, iris_y_train.shape
```
Out[14]:
```
((75, 4), (75,))
```

Ввод [15]:
```
# Размер тестовой выборки
iris_X_test.shape, iris_y_test.shape
```
Out[15]:
```
((75, 4), (75,))
```

Функция train_test_split разделила исходную выборку таким образом, чтобы в обучающей и тестовой частях сохранились все классы.

Ввод [16]:
```
np.unique(iris_y_train)
```
Out[16]:
```
array([0, 1, 2])
```

Ввод [17]:
```
np.unique(iris_y_test)
```

Out[17]:
```
array([0, 1, 2])
```

Построим базовые модели на основе метода ближайших соседей

Ввод [18]:
```
# 2 ближайших соседа
cl1_1 = KNeighborsClassifier(n_neighbors=2)
cl1_1.fit(iris_X_train, iris_y_train)
target1_1 = cl1_1.predict(iris_X_test)
len(target1_1), target1_1
```

Out[18]:
```
(75,
 array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        1, 0, 2, 1, 0, 0, 1, 1, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,
        1, 2, 0, 0, 0, 1, 0, 0, 2, 2, 2, 2, 1, 1, 2, 1, 0, 2, 1, 0, 0, 2,
        0, 1, 2, 1, 1, 2, 1, 0, 1]))
```

Ввод [19]:
```
# 10 ближайших соседей
cl1_2 = KNeighborsClassifier(n_neighbors=10)
cl1_2.fit(iris_X_train, iris_y_train)
target1_2 = cl1_2.predict(iris_X_test)
len(target1_2), target1_2
```

Out[19]:
```
(75,
 array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,
        2, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,
        1, 2, 0, 0, 0, 1, 0, 0, 2, 2, 2, 2, 2, 1, 2, 1, 0, 2, 2, 0, 0, 2,
        0, 2, 2, 1, 1, 2, 2, 0, 1]))
```

Метрики качества классификации

Accuracy

Метрика вычисляет процент (долю в диапазоне от 0 до 1) правильно определенных классов.

Ввод [20]:
```
# iris_y_test - эталонное значение классов из исходной (тестовой) выборки
# target* - предсказанное значение классов

# 2 ближайших соседа
accuracy_score(iris_y_test, target1_1)
```

Out[20]:
```
0.92
```

Ввод [21]:
```
# 10 ближайших соседей
accuracy_score(iris_y_test, target1_2)
```

Out[21]:
```
0.9733333333333334
```

Ввод [22]:
```
# Функция для вычисления метрики accuracy для каждого класса
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    classes = np.unique(y_true)
    res = dict()
    for c in classes:
        temp_data_flt = df[df['t']==c]
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        res[c] = temp_acc
    return res
def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    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]))
```

Ввод [23]:
```
# 2 ближайших соседа
print_accuracy_score_for_classes(iris_y_test, target1_1)
```

```
Метка    Accuracy
0        1.0
1        1.0
2        0.7777777777777778
```

Ввод [24]:
```
# 10 ближайших соседей
print_accuracy_score_for_classes(iris_y_test, target1_2)
```

```
Метка    Accuracy
0        1.0
1        0.9583333333333334
2        0.9629629629629629
```

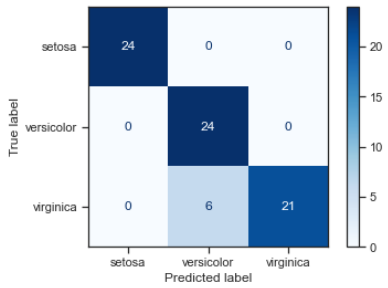
Ввод [25]:
```
# Матрица ошибок или Confusion Matrix
confusion_matrix(iris_y_test, target1_1, labels=[0, 1, 2])
```

Out[25]:
```
array([[24,  0,  0],
       [ 0, 24,  0],
       [ 0,  6, 21]], dtype=int64)
```

Ввод [26]:
```
plot_confusion_matrix(cl1_1, iris_X_test, iris_y_test,
                      display_labels=iris.target_names, cmap=plt.cm.Blues)
```

```
C:\Users\Truma\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Functi
on plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use o
ne of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

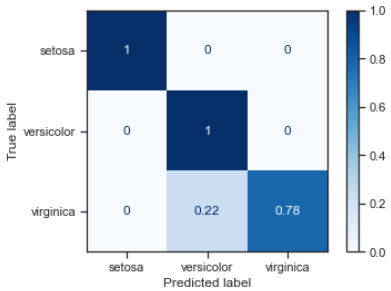
Out[26]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x18fe6936da0>
```



Ввод [27]:
```
plot_confusion_matrix(cl1_1, iris_X_test, iris_y_test,
                      display_labels=iris.target_names,
                      cmap=plt.cm.Blues, normalize='true')
```

```
C:\Users\Truma\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Functi
on plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use o
ne of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```

Out[27]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x18fe6a6e6b0>
```



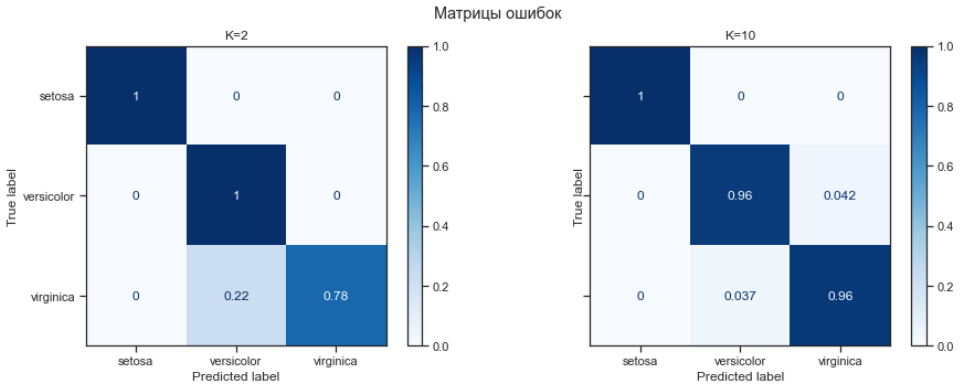
Ввод [28]:
```
fig, ax = plt.subplots(1, 2, sharex='col', sharey='row', figsize=(15,5))

plot_confusion_matrix(cl1_1, iris_X_test, iris_y_test,
                      display_labels=iris.target_names,
                      cmap=plt.cm.Blues, normalize='true', ax=ax[0])

plot_confusion_matrix(cl1_2, iris_X_test, iris_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\Truma\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Functi
on plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use o
ne of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
C:\Users\Truma\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Functi
on plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use o
ne of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```



Precision и F-мера

Precision

Ввод [29]:
```
# Параметры TP, TN, FP, FN считаются как сумма по всем классам
precision_score(iris_y_test, target1_1, average='micro')
```

Out[29]: 0.92

Ввод [30]:
```
# Параметры TP, TN, FP, FN считаются отдельно для каждого класса
# и берется среднее значение, дисбаланс классов не учитывается.
precision_score(iris_y_test, target1_1, average='macro')

```

Out[30]: 0.9333333333333332

Ввод [31]:
```
# 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(iris_y_test, target1_1, average='weighted')
```

Out[31]: 0.936

F-мера

Ввод [32]:
```
f1_score(iris_y_test, target1_1, average='micro')
```

Out[32]:
```
0.92
```

Ввод [33]:
```
f1_score(iris_y_test, target1_1, average='macro')
```

Out[33]:
```
0.9212962962962963
```

Ввод [34]:
```
f1_score(iris_y_test, target1_1, average='weighted')
```

Out[34]:
```
0.9194444444444446
```

Функция classification_report позволяет выводить значения точности, полноты и F-меры для всех классов выборки.

Ввод [35]:
```
classification_report(iris_y_test, target1_1,
                      target_names=iris.target_names, output_dict=True)
```

Out[35]:
```
{'setosa': {'precision': 1.0, 'recall': 1.0, 'f1-score': 1.0, 'support': 24},
 'versicolor': {'precision': 0.8,
  'recall': 1.0,
  'f1-score': 0.888888888888889,
  'support': 24},
 'virginica': {'precision': 1.0,
  'recall': 0.7777777777777778,
  'f1-score': 0.8750000000000001,
  'support': 27},
 'accuracy': 0.92,
 'macro avg': {'precision': 0.9333333333333332,
  'recall': 0.9259259259259259,
  'f1-score': 0.9212962962962963,
  'support': 75},
 'weighted avg': {'precision': 0.936,
  'recall': 0.92,
  'f1-score': 0.9194444444444446,
  'support': 75}}
```

Оптимизация гиперпараметров

Grid Search (решетчатый поиск) ¶

Ввод [36]:
```
n_range = np.array(range(5,55,5))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[36]:
```
[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}]
```

Ввод [37]:
```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accuracy')
clf_gs.fit(iris_X_train, iris_y_train)
```

```
CPU times: total: 78.1 ms
Wall time: 77.8 ms
```

Out[37]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}],
             scoring='accuracy')
```

Ввод [38]:
```
clf_gs.cv_results_
```

Out[38]:
```
{'mean_fit_time': array([0.00039859, 0.00039902, 0.00059843, 0.00059848, 0.00039892,
       0.00039897, 0.00039883, 0.00039892, 0.00039883, 0.00019927]),
 'std_fit_time': array([0.00048817, 0.00048869, 0.00048862, 0.00048866, 0.00048858,
       0.00048864, 0.00048846, 0.00048858, 0.00048846, 0.00039854]),
 'mean_score_time': array([0.0009975 , 0.0007978 , 0.00099721, 0.0009973 , 0.0009973 ,
       0.00059829, 0.00079789, 0.00099745, 0.00059838, 0.00099759]),
 'std_score_time': array([1.78416128e-07, 3.98898363e-04, 4.67203091e-07, 1.50789149e-07,
       2.61174468e-07, 4.88499866e-04, 3.98945848e-04, 2.43140197e-07,
       4.88577702e-04, 4.10190833e-07]),
 '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([0.93333333, 0.93333333, 0.93333333, 1.        , 1.        ,
       0.93333333, 1.        , 0.66666667, 0.66666667, 0.66666667]),
 'split1_test_score': array([1.        , 1.        , 1.        , 1.        , 0.93333333,
       1.        , 0.93333333, 0.66666667, 0.66666667, 0.66666667]),
 'split2_test_score': array([1.        , 1.        , 0.93333333, 1.        , 0.93333333,
       0.86666667, 0.86666667, 0.66666667, 0.66666667, 0.66666667]),
 'split3_test_score': array([0.93333333, 0.93333333, 0.93333333, 0.93333333, 0.93333333,
       0.86666667, 0.86666667, 0.6       , 0.66666667, 0.6       ]),
 'split4_test_score': array([1.        , 1.        , 0.93333333, 0.93333333, 0.93333333,
       0.93333333, 0.93333333, 0.73333333, 0.53333333, 0.33333333]),
 'mean_test_score': array([0.97333333, 0.97333333, 0.94666667, 0.97333333, 0.94666667,
       0.92      , 0.92      , 0.66666667, 0.64      , 0.58666667]),
 'std_test_score': array([0.03265986, 0.03265986, 0.02666667, 0.03265986, 0.02666667,
       0.04988877, 0.04988877, 0.0421637 , 0.05333333, 0.12927146]),
 'rank_test_score': array([ 1,  1,  4,  1,  4,  6,  6,  8,  9, 10])}
```

Ввод [39]:
```
# Лучшая модель
clf_gs.best_estimator_
```

Out[39]:
```
KNeighborsClassifier()
```

Ввод [40]:
```
# Лучшее значение метрики
clf_gs.best_score_
```

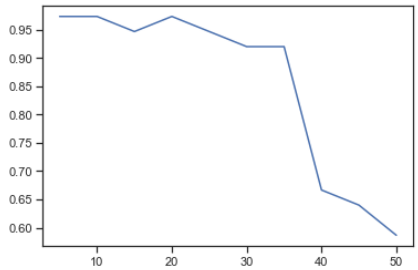
Out[40]:
```
0.9733333333333334
```

Ввод [41]:
```
# Лучшее значение параметров
clf_gs.best_params_
```

Out[41]:
```
{'n_neighbors': 5}
```

Ввод [42]:
```
# Изменение качества на тестовой выборке в зависимости от K-соседей
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'])
```

Out[42]:
```
[<matplotlib.lines.Line2D at 0x18fe8d8eb60>]
```



Randomized Search (случайный поиск)

Используется в том случае когда полный решетчатый поиск работает слишком долго.

Ввод [43]:
```
%%time
clf_rs = RandomizedSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='accuracy')
clf_rs.fit(iris_X_train, iris_y_train)
```

```
CPU times: total: 78.1 ms
Wall time: 74.8 ms
```

Out[43]:
```
RandomizedSearchCV(cv=5, estimator=KNeighborsClassifier(),
                   param_distributions=[{'n_neighbors': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])}],
                   scoring='accuracy')
```

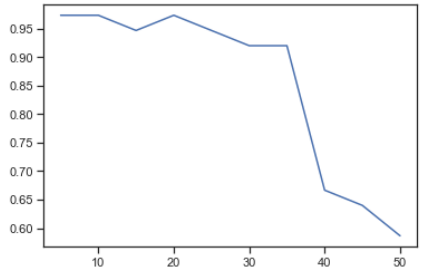
Ввод [44]:
```
# В данном случае оба способа нашли одинаковое решение
clf_rs.best_score_, clf_rs.best_params_
```

Out[44]:
```
(0.9733333333333334, {'n_neighbors': 5})
```

Ввод [45]:
```
clf_gs.best_score_, clf_gs.best_params_
```

Out[45]:
```
(0.9733333333333334, {'n_neighbors': 5})
```

Out[46]:
```
[<matplotlib.lines.Line2D at 0x18fe8e066e0>]
```



Построение кривых обучения и валидации

Построение кривых обучения - learning_curve

Строится зависимость метрики на обучающей выборке от размера выборки.

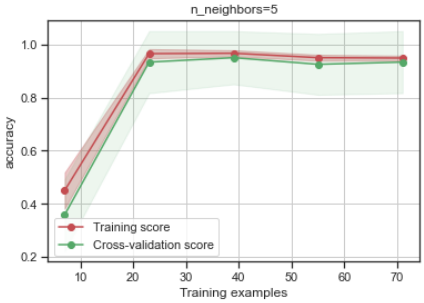
Ввод [47]:
```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0, 5), scoring='accuracy'):
    plt.figure()
    plt.title(title)
    if ylim is not None:
        plt.ylim(*ylim)
    plt.xlabel("Training examples")
    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
```

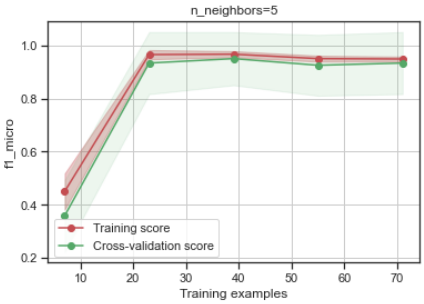
Ввод [48]:
```
plot_learning_curve(KNeighborsClassifier(n_neighbors=5), 'n_neighbors=5',
                    iris_X_train, iris_y_train, cv=20)
```

Out[48]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\Truma\\AppData\\Local\\Programs\\Python\\Python310\\lib\\site-packages\\matplotlib
\\pyplot.py'>
```



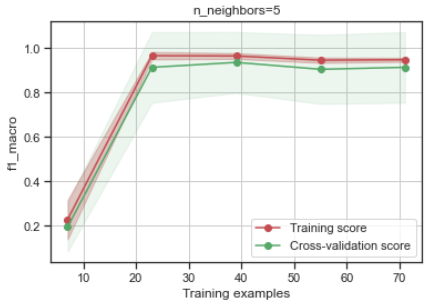
Ввод [49]:
```
plot_learning_curve(KNeighborsClassifier(n_neighbors=5), 'n_neighbors=5',
                    iris_X_train, iris_y_train, cv=20, scoring='f1_micro')
```

Out[49]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\Truma\\AppData\\Local\\Programs\\Python\\Python310\\lib\\site-packages\\matplotlib
\\pyplot.py'>
```



Ввод [50]:
```
plot_learning_curve(KNeighborsClassifier(n_neighbors=5), 'n_neighbors=5',
                    iris_X_train, iris_y_train, cv=20, scoring='f1_macro')
```

Out[50]:
```
<module 'matplotlib.pyplot' from 'C:\\Users\\Truma\\AppData\\Local\\Programs\\Python\\Python310\\lib\\site-packages\\matplotlib
\\pyplot.py'>
```



Построение кривой валидации - validation_curve

Строится зависимость метрики на тестовой выборке от одного из гиперпараметров.

Ввод [51]:

```
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
```

Ввод [52]:

```
plot_validation_curve(KNeighborsClassifier(), 'knn',
                      iris_X_train, iris_y_train,
                      param_name='n_neighbors', param_range=n_range,
                      cv=20, scoring="accuracy")
```

Out[52]:

```
<module 'matplotlib.pyplot' from 'C:\\Users\\Truma\\AppData\\Local\\Programs\\Python\\Python310\\lib\\site-packages\\matplotlib
\\pyplot.py'>
```