Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

ОТЧЁТ ПО

Лабораторной работе №3

Выполнил:
студент группы ИУ5-65Б
Кулешова И. А.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.
Подпись и дата:

Москва

2024

1) Описание задания:

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра K. Оцените качество модели с помощью подходящих для задачи метрик.
5. Произведите подбор гиперпараметра K с использованием GridSearchCV и RandomizedSearchCV и кросс-валидации, оцените качество оптимальной модели. Используйте не менее двух стратегий кросс-валидации.
6. Сравните метрики качества исходной и оптимальной моделей.

2) Текст программы и итоги:

```
!pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\user\
anaconda3\lib\site-packages (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\user\
anaconda3\lib\site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.3.2 in c:\users\user\
anaconda3\lib\site-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in c:\users\user\
anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\user\
anaconda3\lib\site-packages (from scikit-learn) (2.2.0)
```

```
import numpy as np
import pandas as pd
import sklearn
from typing import Dict, Tuple
from scipy import stats
from sklearn import datasets
from sklearn import model_selection
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 import metrics
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")
```

Загрузка и первичный анализ данных

```
data = pd.read_csv('data/onlinefoods.csv', sep=",")

# размер набора данных
data.shape
```

```
(388, 13)
```

```
# типы колонок
data.dtypes
```

```
Age                                int64
Gender                            object
```

```
Marital Status                  object
Occupation                      object
Monthly Income                  object
Educational Qualifications      object
Family size                      int64
latitude                       float64
longitude                      float64
Pin code                         int64
Output                          object
Feedback                        object
Unnamed: 12                     object
dtype: object
```

```
# проверим есть ли пропущенные значения
data.isnull().sum()
```

```
Age                           0
Gender                        0
Marital Status                0
Occupation                    0
Monthly Income                0
Educational Qualifications    0
Family size                   0
latitude                      0
longitude                     0
Pin code                      0
Output                        0
Feedback                      0
Unnamed: 12                   0
dtype: int64
```

```
# Первые 5 строк датасета
data.head()
```

```
   Age  Gender Marital Status Occupation  Monthly Income  \
0   20  Female         Single    Student       No Income
1   24  Female         Single    Student  Below Rs.10000
2   22    Male         Single    Student  Below Rs.10000
3   22  Female         Single    Student       No Income
4   22    Male         Single    Student  Below Rs.10000

  Educational Qualifications  Family size  latitude  longitude  Pin
code  \
0              Post Graduate            4   12.9766    77.5993
560001
1                   Graduate            3   12.9770    77.5773
560009
2              Post Graduate            3   12.9551    77.6593
560017
3                   Graduate            6   12.9473    77.5616
```

```
560019
4               Post Graduate            4   12.9850    77.5533
560010

  Output   Feedback Unnamed: 12
0     Yes   Positive         Yes
1     Yes   Positive         Yes
2     Yes  Negative          Yes
3     Yes   Positive         Yes
4     Yes   Positive         Yes
```

```
total_count = data.shape[0]
print('Всего строк: {}'.format(total_count))
```

```
Всего строк: 388
```

Так как пропусков нет, то этап заполнения пропусков можно пропустить.

Кодирование категориальных признаков

```
from sklearn.preprocessing import OrdinalEncoder

data_oe = data[['Family size', 'Feedback']]

oe = OrdinalEncoder()
cat_enc_oe = oe.fit_transform(data_oe)
cat_enc_oe
```

```
array([[3., 1.],
       [2., 1.],
       [2., 0.],
       [5., 1.],
       [3., 1.],
       [1., 1.],
       [2., 1.],
       [2., 1.],
       [1., 1.],
       [3., 1.],
       [4., 1.],
       [1., 0.],
       [4., 1.],
       [3., 1.],
       [4., 1.],
       [5., 1.],
       [1., 1.],
       [2., 0.],
       [3., 0.],
       [0., 1.],
       [2., 1.],
       [3., 1.],
```

```
       [3., 1.],
       [3., 1.],
       [2., 1.],
       [2., 1.],
       [4., 1.],
       [2., 1.],
       [2., 1.],
       [3., 1.],
       [4., 1.],
       [3., 1.],
       [3., 1.],
       [4., 1.],
       [1., 1.],
       [2., 1.],
       [4., 1.],
       [4., 0.],
       [2., 1.],
       [3., 1.],
       [3., 0.],
       [2., 1.],
       [3., 1.],
       [4., 1.],
       [4., 1.],
       [1., 1.],
       [2., 1.],
       [1., 1.],
       [2., 0.],
       [4., 1.],
       [4., 1.],
       [1., 1.],
       [2., 1.],
       [1., 1.],
       [2., 1.],
       [1., 1.],
       [0., 1.],
       [2., 1.],
       [4., 1.],
       [3., 1.],
       [4., 1.],
       [2., 1.],
       [0., 1.],
       [3., 1.],
       [1., 1.],
       [5., 1.],
       [3., 1.],
       [3., 1.],
       [3., 1.],
       [3., 1.],
       [3., 1.],
```

```
       [2., 1.],
       [2., 1.],
       [1., 1.],
       [3., 1.],
       [2., 1.],
       [3., 1.],
       [3., 1.],
       [2., 1.],
       [4., 1.],
       [3., 1.],
       [2., 1.],
       [2., 1.],
       [2., 1.],
       [2., 1.],
       [3., 1.],
       [4., 1.],
       [0., 1.],
       [4., 1.],
       [1., 0.],
       [4., 1.],
       [0., 1.],
       [1., 1.],
       [2., 1.],
       [0., 1.],
       [0., 1.],
       [4., 1.],
       [4., 1.],
       [4., 1.],
       [2., 1.],
       [2., 1.],
       [1., 1.],
       [2., 1.],
       [2., 1.],
       [1., 0.],
       [2., 1.],
       [1., 1.],
       [0., 1.],
       [1., 1.],
       [4., 1.],
       [3., 1.],
       [0., 1.],
       [1., 1.],
       [4., 1.],
       [1., 1.],
       [3., 0.],
       [4., 1.],
       [1., 1.],
       [4., 1.],
       [2., 1.],
```

```
       [4., 1.],
       [2., 1.],
       [2., 1.],
       [4., 0.],
       [5., 0.],
       [2., 1.],
       [2., 1.],
       [3., 1.],
       [2., 1.],
       [3., 1.],
       [2., 1.],
       [0., 1.],
       [1., 1.],
       [1., 0.],
       [3., 1.],
       [1., 1.],
       [2., 1.],
       [4., 0.],
       [2., 1.],
       [3., 1.],
       [2., 1.],
       [2., 0.],
       [1., 1.],
       [4., 1.],
       [2., 0.],
       [2., 1.],
       [2., 1.],
       [3., 1.],
       [3., 1.],
       [3., 1.],
       [2., 1.],
       [3., 1.],
       [1., 1.],
       [1., 1.],
       [4., 1.],
       [3., 1.],
       [2., 1.],
       [2., 1.],
       [4., 0.],
       [1., 1.],
       [1., 1.],
       [2., 0.],
       [3., 1.],
       [2., 1.],
       [2., 1.],
       [3., 1.],
       [4., 0.],
       [4., 1.],
       [1., 1.],
```

```
       [1., 0.],
       [0., 1.],
       [2., 1.],
       [0., 1.],
       [2., 1.],
       [1., 1.],
       [5., 1.],
       [1., 1.],
       [2., 0.],
       [0., 0.],
       [2., 1.],
       [2., 1.],
       [2., 0.],
       [3., 1.],
       [1., 1.],
       [4., 1.],
       [0., 1.],
       [1., 1.],
       [1., 1.],
       [1., 0.],
       [5., 1.],
       [2., 0.],
       [1., 1.],
       [4., 1.],
       [1., 1.],
       [1., 1.],
       [2., 1.],
       [5., 1.],
       [5., 1.],
       [5., 1.],
       [1., 0.],
       [1., 1.],
       [4., 1.],
       [3., 1.],
       [3., 1.],
       [2., 1.],
       [1., 1.],
       [3., 0.],
       [0., 1.],
       [1., 1.],
       [3., 0.],
       [5., 0.],
       [2., 1.],
       [1., 0.],
       [2., 1.],
       [3., 1.],
       [2., 0.],
       [4., 1.],
       [1., 0.],
```

```
       [4., 1.],
       [2., 1.],
       [0., 1.],
       [1., 1.],
       [3., 1.],
       [4., 1.],
       [1., 1.],
       [1., 1.],
       [2., 1.],
       [1., 1.],
       [4., 0.],
       [2., 1.],
       [5., 1.],
       [4., 1.],
       [2., 0.],
       [1., 1.],
       [2., 1.],
       [2., 0.],
       [2., 1.],
       [1., 1.],
       [1., 1.],
       [2., 1.],
       [5., 1.],
       [1., 1.],
       [2., 1.],
       [1., 1.],
       [0., 0.],
       [2., 1.],
       [5., 0.],
       [2., 1.],
       [0., 0.],
       [1., 1.],
       [2., 1.],
       [5., 1.],
       [2., 1.],
       [1., 1.],
       [5., 1.],
       [2., 1.],
       [1., 0.],
       [5., 0.],
       [2., 1.],
       [5., 0.],
       [5., 1.],
       [1., 0.],
       [1., 0.],
       [2., 0.],
       [3., 1.],
       [1., 1.],
       [2., 0.],
```

```
        [1., 1.],
        [2., 1.],
        [1., 1.],
        [1., 1.],
        [1., 0.],
        [2., 1.],
        [1., 1.],
        [1., 1.],
        [3., 1.],
        [2., 1.],
        [1., 1.],
        [4., 0.],
        [3., 1.],
        [5., 1.],
        [1., 1.],
        [3., 1.],
        [2., 1.],
        [1., 1.],
        [1., 1.],
        [1., 1.],
        [4., 0.],
        [2., 1.],
        [1., 1.],
        [5., 1.],
        [2., 1.],
        [4., 0.],
        [0., 0.],
        [2., 0.],
        [4., 1.],
        [2., 1.],
        [0., 1.],
        [1., 0.],
        [5., 0.],
        [2., 0.],
        [5., 1.],
        [3., 1.],
        [1., 1.],
        [2., 1.],
        [2., 1.],
        [1., 1.],
        [3., 1.],
        [4., 1.],
        [1., 0.],
        [4., 1.],
        [1., 1.],
        [1., 1.],
        [2., 1.],
        [1., 1.],
        [4., 0.],
```

```
       [2., 1.],
       [5., 1.],
       [4., 1.],
       [2., 0.],
       [1., 1.],
       [2., 1.],
       [2., 0.],
       [2., 1.],
       [1., 1.],
       [1., 1.],
       [2., 1.],
       [5., 1.],
       [1., 1.],
       [3., 1.],
       [3., 1.],
       [2., 1.],
       [1., 1.],
       [3., 0.],
       [0., 1.],
       [1., 1.],
       [3., 0.],
       [5., 1.],
       [2., 1.],
       [1., 0.],
       [2., 1.],
       [3., 1.],
       [2., 0.],
       [4., 1.],
       [1., 0.],
       [3., 1.],
       [3., 1.],
       [2., 1.],
       [1., 1.],
       [3., 0.],
       [0., 1.],
       [1., 1.],
       [3., 1.],
       [5., 1.],
       [2., 1.],
       [1., 1.],
       [1., 1.],
       [4., 0.],
       [2., 1.],
       [5., 1.],
       [4., 1.],
       [2., 0.],
       [1., 1.],
       [2., 1.],
       [2., 0.],
```

```
       [2., 1.],
       [1., 1.],
       [1., 1.],
       [2., 1.],
       [5., 1.],
       [1., 1.],
       [2., 1.],
       [1., 1.],
       [0., 0.],
       [2., 0.],
       [5., 0.],
       [2., 1.],
       [0., 0.],
       [1., 1.],
       [3., 1.],
       [1., 1.],
       [2., 1.],
       [2., 1.],
       [1., 1.],
       [3., 1.],
       [4., 1.],
       [1., 1.],
       [4., 1.]])
```

```
# Уникальные значения 1 признака
np.unique(cat_enc_oe[:, 0])
```

```
array([0., 1., 2., 3., 4., 5.])
```

```
# Уникальные значения 2 признака
np.unique(cat_enc_oe[:, 1])
```

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

Разделение выборки на обучающую и тестовую

```
data= np.c_[cat_enc_oe[:, 0], cat_enc_oe[:, 1]]

data_x_train, data_x_test, data_y_train, data_y_test =
train_test_split(data, cat_enc_oe[:, 1], test_size=0.2,
random_state=1)

# Размер обучающей выборки
data_x_train.shape, data_y_train.shape
```

```
((310, 2), (310,))
```

```
# Размер тестовой выборки
data_x_test.shape, data_y_test.shape
```

```
((78, 2), (78,))
```

```
np.unique(data_y_train)
```

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

```
np.unique(data_y_test)
```

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

```
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))
```

```
# В исходной выборке нет явного дисбаланса классов для целевого 
признака
print_class_proportions(cat_enc_oe[:, 1])
```

```
Метка 	 Количество 	 Процент встречаемости
0.0 	 71 	 	 18.3%
1.0 	 317 	 	 81.7%
```

```
# Для обучающей выборки
print_class_proportions(data_y_train)
```

```
Метка        Количество        Процент встречаемости
0.0    58          18.71%
1.0    252         81.29%
```

```
# Для тестовой выборки
print_class_proportions(data_y_test)
```

```
Метка        Количество        Процент встречаемости
0.0    13          16.67%
1.0    65          83.33%
```

Модель ближайших соседей для произвольно заданного гиперпараметра K. Оценка качества модели с помощью подходящих для задачи метрик.

```
# 27 ближайших соседей
cl1_1 = KNeighborsClassifier(n_neighbors=27)
cl1_1.fit(data_x_train, data_y_train)
```

```
KNeighborsClassifier(n_neighbors=27)
```

```
target1_1 = cl1_1.predict(data_x_test)
len(target1_1), target1_1
```

```
(78,
 array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
1., 1.,
        1., 1., 1., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1.,
1., 1.,
        1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 1.,
1., 1.,
        1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1.,
1., 1.,
        1., 1., 1., 1., 1., 1., 1., 1., 1., 0.]))
```

Так как класс не сбалансирован, то будем использовать метрику Precision, recall и F-мера для оценки качества модели.

```
precision_score(data_y_test, target1_1), recall_score(data_y_test,
target1_1)
```

```
(0.9420289855072463, 1.0)
```

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

```
0.9516908212560387
```

```
f1_score(data_y_test, target1_1, average='weighted')
```

```
0.9448213478064225
```

Вывод: качество модели высокое.

Подбор гиперпараметра K

```
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
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

n_range = np.array([1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2,
3, 4, 5, 1, 2, 3, 4, 5])
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
[{'n_neighbors': array([1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
1, 2, 3, 4, 5, 1, 2,
         3, 4, 5])}]
```

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=7,
scoring='accuracy')
clf_gs.fit(data_x_train, data_y_train)
```

```
CPU times: total: 812 ms
Wall time: 821 ms
```

```
GridSearchCV(cv=7, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([1, 2, 3, 4, 5, 1, 2,
3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2,
       3, 4, 5])}],
             scoring='accuracy')
```

```
clf_gs.cv_results_
```

```
{'mean_fit_time': array([0.0008488 , 0.00089312, 0.00068358,
0.00074404, 0.00071229,
```

```
        0.00057013, 0.00085493, 0.00073515, 0.00057002, 0.00057002,
        0.00056992, 0.00085514, 0.00056999, 0.00057002, 0.00085507,
        0.00056996, 0.00049189, 0.00056747, 0.00042759, 0.00083453,
        0.00071955, 0.00085674, 0.00068392, 0.00056423, 0.00099812]),
 'std_fit_time': array([6.35850677e-04, 3.76105894e-04, 4.34903542e-
04, 4.25540645e-04,
        4.50493906e-04, 4.93744438e-04, 3.49025918e-04, 4.68029245e-
04,
        4.93655942e-04, 4.93655958e-04, 4.93567458e-04, 3.49109231e-
04,
        4.93626451e-04, 4.93655942e-04, 3.49081483e-04, 4.93596973e-
04,
        4.62120587e-04, 4.91477308e-04, 4.93734605e-04, 3.42394830e-
04,
        4.40180252e-04, 3.49793101e-04, 4.14029063e-04, 4.88814405e-
04,
        9.85970826e-07]),
 'mean_score_time': array([0.00512651, 0.00429327, 0.00359055,
0.00353037, 0.00370448,
        0.00359283, 0.00327703, 0.00396664, 0.00354471, 0.0035617 ,
        0.00342911, 0.0034194 , 0.00341896, 0.00356184, 0.00356177,
        0.00369467, 0.00352308, 0.00371984, 0.00370414, 0.00386732,
        0.00370472, 0.00327754, 0.00379617, 0.00328251, 0.00299113]),
 'std_score_time': array([1.11419154e-03, 4.74665409e-04, 5.20446350e-
04, 4.68985091e-04,
        4.50741599e-04, 4.61895317e-04, 4.50956760e-04, 7.25600376e-
04,
        4.80579977e-04, 4.93518288e-04, 4.85528930e-04, 4.94030056e-
04,
        4.93793919e-04, 4.93774340e-04, 4.93577273e-04, 6.94577911e-
04,
        4.93088974e-04, 4.32091327e-04, 4.50601144e-04, 6.42459295e-
04,
        6.34471289e-04, 4.55333762e-04, 6.69032923e-04, 4.44720808e-
04,
        1.64547952e-06]),
 'param_n_neighbors': masked_array(data=[1, 2, 3, 4, 5, 1, 2, 3, 4, 5,
1, 2, 3, 4, 5, 1, 2, 3,
                    4, 5, 1, 2, 3, 4, 5],
              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': 1},
```

```
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5},
  {'n_neighbors': 1},
  {'n_neighbors': 2},
  {'n_neighbors': 3},
  {'n_neighbors': 4},
  {'n_neighbors': 5}],
 'split0_test_score': array([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.]),
 'split1_test_score': array([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.]),
 'split2_test_score': array([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.]),
 'split3_test_score': array([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.]),
 'split4_test_score': array([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.]),
 'split5_test_score': array([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.]),
 'split6_test_score': array([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.]),
 'mean_test_score': array([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.]),
 'std_test_score': 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.]),
 'rank_test_score': array([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])}
```

```
# Лучшая модель
clf_gs.best_estimator_
```

```
KNeighborsClassifier(n_neighbors=1)
```

```
# Лучшее значение метрики
clf_gs.best_score_
```

```
1.0
```

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

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

Делаем то же самое, только с помощью Randomized Search:

```
%%time
clf_rs = RandomizedSearchCV(KNeighborsClassifier(), tuned_parameters,
cv=7, scoring='accuracy')
clf_rs.fit(data_x_train, data_y_train)
```

```
CPU times: total: 344 ms
Wall time: 344 ms
```

```
RandomizedSearchCV(cv=7, estimator=KNeighborsClassifier(),
                   param_distributions=[{'n_neighbors': array([1, 2,
3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2,
       3, 4, 5])}],
                   scoring='accuracy')
```

```
clf_rs.best_score_, clf_rs.best_params_
```

```
(1.0, {'n_neighbors': 5})
```

```
clf_gs.best_score_, clf_gs.best_params_
```

```
(1.0, {'n_neighbors': 1})
```

Используем стратегии StratifiedKFold и StratifiedShuffleSplit кросс-валидации:

```
X = cat_enc_oe[:, 0]
y = cat_enc_oe[:, 1]
skf = StratifiedKFold(n_splits=3)
```

```python
for train, test in skf.split(X, y):
    print("%s %s" % (train, test))
```

```
[116 117 118 119 120 121 122 125 126 127 128 129 130 131 132 134 135
136
 138 139 140 142 143 145 146 147 148 149 150 151 152 153 154 155 156
157
 159 160 162 163 164 165 167 168 170 171 172 173 174 175 176 179 180
182
 183 184 185 186 187 189 190 191 192 193 194 195 196 197 198 199 200
201
 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218
219
 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236
237
 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254
255
 256 257 258 259 260 261 262 263 264 265 266 267 268 269 270 271 272
273
 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290
291
 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308
309
 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326
327
 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344
345
 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362
363
 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380
381
 382 383 384 385 386 387] [  0   1   2   3   4   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  30  31  32  33  34
35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70
71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88
89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106
107
 108 109 110 111 112 113 114 115 123 124 133 137 141 144 158 161 166
169
 177 178 181 188]
[  0   1   2   3   4   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  30  31  32  33  34
35
```

```
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70
71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88
89
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106
107
 108 109 110 111 112 113 114 115 123 124 133 137 141 144 158 161 166
169
 177 178 181 188 250 251 252 253 254 255 258 260 264 265 267 268 269
270
 272 273 274 275 276 277 279 280 281 282 283 284 285 286 287 288 289
290
 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307
308
 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325
326
 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343
344
 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361
362
 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379
380
 381 382 383 384 385 386 387] [116 117 118 119 120 121 122 125 126 127
128 129 130 131 132 134 135 136
 138 139 140 142 143 145 146 147 148 149 150 151 152 153 154 155 156
157
 159 160 162 163 164 165 167 168 170 171 172 173 174 175 176 179 180
182
 183 184 185 186 187 189 190 191 192 193 194 195 196 197 198 199 200
201
 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218
219
 220 221 222 223 224 225 226 227 228 229 230 231 232 233 234 235 236
237
 238 239 240 241 242 243 244 245 246 247 248 249 256 257 259 261 262
263
 266 271 278]
[  0   1   2   3   4   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  30  31  32  33  34
35
  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52
53
  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70
71
  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88
89
```

```
  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106
107
 108 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124
125
 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142
143
 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
161
 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178
179
 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196
197
 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214
215
 216 217 218 219 220 221 222 223 224 225 226 227 228 229 230 231 232
233
 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 256
257
 259 261 262 263 266 271 278] [250 251 252 253 254 255 258 260 264 265
267 268 269 270 272 273 274 275
 276 277 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293
294
 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311
312
 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329
330
 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347
348
 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365
366
 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383
384
 385 386 387]
```

```
from sklearn.model_selection import StratifiedShuffleSplit
X = cat_enc_oe
y = y = cat_enc_oe[:, 1]
sss = StratifiedShuffleSplit(n_splits=5, random_state=0)
for i, (train_index, test_index) in enumerate(sss.split(X, y)):
     print(f"Fold {i}:")
     print(f"  Train: {train_index}")
     print(f"  Test:  {test_index}")
```

```
Fold 0:
  Train: [170 375 168 256 196  83 117 370 166   1 173 223 262  28 232
295 257 385
 355 235  67 254  64 222 224  43  59 341  95 210 300 332 237  45 133
209
 298 275 259 272 285 193 100 105   2 169 342   3 334  77 163 270  63
4
```

```
 197 226 271  10  31  94 183 371  16  37 158 155 255  70 164 174 343
87
 335 132 248 128 228  36 244 205 314 319 247 292 377 291   6  34 233
176
  73 144 265 200 276  85  25 157  47 156  88 211 199 337  46 120 212
125
 137 281 221 358 263 145  75 316 127 324 339  69 326  23 250 320  15
153
 331 347 351 288 165 106 202 253  44 333 367   9 220 344 191  96 112
79
  35  11 179  33 296 177 323  48 171 304 139 297 130 284  24 225 245
198
 294  18 307 374 369 308  91 109  86  20 325 338 234  29 327 301 239
359
 123  74 313 380 353  39 252 283 141 329 149   8 366  32 214 378 119
352
  98 217 185  13 121 274 302 360 136 286 182  19 162 189 143 172 269
184
 278 218 241 315  41 381  89  90 159 216 365  84  65 345 104  21 190
140
  57  80 161 111   7 321 303  17 148   5 110  93  72 208  78  61 116
97
 290 229 310 267 268  56 328 126 249 150 348  26 192  81 322 152 379
363
  68 317 180 299 356 206 364 138 306  42  12 135  51 264 372 251  30
38
 280 384 154 386  52 282  62 195 387 231 260 309 243 289 122 181  71
354
 383 124 240 346 201 312 118  53 258 175 305  22 188  40 146 114 277
82
 361 131 227 213 101 151 187  14 368  66 336 382 362 261 203 113 330
349
 293 376 350 215 129 279  58]
  Test:  [ 27 115 194  92 204 178 236 103  50 373 238  99 107 142  76
160 266 207
 230  60 340 242 273 357 287 186  55 219 167 246 147  49   0 134 102
318
  54 108 311]
Fold 1:
  Train: [ 11  74  62  30 242   6 335 230  97   9 152 130 199 103 325
31 237 196
 315  75  68  96 357 356 253 372 100 297  89 262  71 265  44 157 260
247
 175 250 257 203 351 238  27  21  82 219  38 201 102  41 368 159 222
106
 284 155 333 124 228 264 174 136 282  37 123 172  51 113  42 186 328
227
  92 316   5 119 132 134 332 353 323  59 163 302  45 236 366  46 350
279
```

```
 349 168  23  26 320 381 187 365 273 307 327 339  24 217 188 329 167
189
 361  61 224 169 209 183 343  12 212 164 334 165 296 290 346 133  58
33
 360 114 178 118  65 383 220 221 143  69  20   4 141  81  53  17 135
60
  48 375  64 166 140 144 278  99 266 142 289  29  77 313 314 252  40
281
 239 272 151 258 259  49  86 245 180 204 379 117 342 149  14 207 251
127
  39  93 301 354 263 231 202 275 193 173 283 223 261 300  16 148 277
269
 116 138 153  19 267 177  43 274  78 184  88 235 176 154   7 312  35
385
 359 248 214 308 126 306  80  28 200 194 170 101 162 299 213  63 246
305
 128 386  36  52 370 108 226 374 156 304  91 249  18 270 285 145 362
10
 303 197  98  76  47 271 295 171 218  70 104  66  34 287 364 122 373
348
 121 382 198 384 298 195 225 336 378  54 318 293 160 129  57 191 215
87
 369  79 326  15 111 112  90 341 311 376 280 276 206  73  32  25 288
208
  67 190 241 347 107 125 232 147 371  83 146 240 158 185 243 137 331
268
 205 229 256   2 216 294 310 139   8 244  94   0 355 210 337   1 309
291
  22 181 352  85 182  56  84]
  Test:  [380 255  50 150 317 387 120 254  95 131 324 233 344   3 358
367 211 330
 377 292 363 319 345 179 161 338 340 109 234 192 110 105 286 322  72
115
  55  13 321]
Fold 2:
  Train: [318 253 174  19  56  69 128 242  15 116 216 331  76 204 366
200 371 104
  66  98 244 148 218 224  28 156  51 208 123   6 107 356 319 122 351
330
 338 166 105 339 149 277 316 364 160  34 154 213 102 147 369 233 377
101
 301  22 378 275   4 281 315 173 299 266 219 214   5 192 383  78  10
343
 259 381 114 385 187 133 265 267 115 239  46 380  44 188  90 229 284
73
  18  12 263 113 195 361 308 370 183  13 269 386 276 294 335 202 283
118
 137 300 176 282 292  60  62 305 363 217 241 142 136 179 120  75 285
59
```

```
  67 121 296  85 279 141 211 111 324 252 298 210 365  61   1 270 209
43
 250  86 307 157 323  49 139 337   0 185  91 199 236  83 189 190  54
238
 329 288 119 373 367 345 326 274 138 286 254 140 163 237 177 240 382
23
 387  99 248 359  71 320 168 321 180 347 354 374 108 334 221 249 222
310
 212 196 360  64 235 289  95 303 197  26 124  38 161 340 309  31 150
96
  82 306 178  35 198 186   9 349 293  97 117 341 336 205 287  37  20
181
 290 272 146 313 130 145  89 194  84  30  17  93 295 379 328 151   7
230
 291 246 297  80   3 103 110 143 260 350 368  39 127  94 175 191 232
106
 182 226  58 203 247 251  65 258 264 215 317 162 201 357  21 372 271
311
 220 126 327 243 255  77  55 353 256 332  88 131  81 262 167 206 155
74
 280 109 384  32 355   2 333 207 125  36  72 132 257 152  52 184  47
223
 362  11 346 302  24  87 144 231  63 153 228 342 312 135 234  27  48
376
  92  50 314 170  45 348 278]
  Test:  [134 261 268 304 129 100  70  29 227  40  68 245 171  14  42
358 325 193
  41 159 375  53  16 158  33 165 322  79 352  25 172 225  57 344 169
164
   8 273 112]
Fold 3:
  Train: [247 111   1 303 279 152 145  86  49 286 299 261 284  89 324
98  78 150
 137 100  80 231  24 327  88 384 330 318 113 162 267 340  28 123 151
82
 357 268 141 169 172  34 229  50 271 353  56 134 346  99 127 226  36
201
 214 341 195 235 115 158 207 280  95 239  22 251 120 217 143  27 379
139
  65 104 227  91 289  66 257 249 140  48 164 177 218 193 276 352 310
347
   5 148 375  67 326 262 336 185 209 339 133 147  60 386 160  11 248
183
  33 266 190 273 308   4 293  90 126  62 360 132  69  12 377 301 254
97
 205 294 188 328 283 309 182 252  25 105 385 297 165 211  75  20   8
156
  61  16  18 298 149 269 237 168 125 378  37 323 202 128 170 292 366
74
```

```
 258 203 204 243 312 245 371 295 259 225 215 191 192  43  77 186 334
373
 109  87  30 197 118 343 219 167 381 354  63  96 103 144  21 241 287
117
 224 344 116  76 210  10 355 382  42  52 321 129 189 364   9 223 122
359
 367 342 338 350 250 345   6  29 329 228 314 107  35 365 136  59 300
3
 376  44 101  51 138  72 270 244  55 199 306 263 315  81 374 368 277
176
  53 212 130 121 372  19 114 155 157 274 351 264  17 184 194  79 331
40
 380 216 304 370 302  46  73 320  54 337 108   7 369 265 163 363  39
317
 291 362 232  57 171 221 208 272 110  38 383   0 187 322  23  45 260
175
 230  26 166 307 319 335 313 124  41 161 238 255  64 174 142 173 135
240
 311 181 356 256  68 325 196  47  93 333 305 233  13 222 112 281  71
296
 102 213  32 348 159  94  92]
  Test:  [  2 288 178 387 275 200 180 198  84 153 234  70 290 206 179
83  85  14
 246 154 285 131 316 146 236 253 282  58 361  15  31 119 332 106 278
358
 220 242 349]
Fold 4:
  Train: [ 89 334 236 359 114  39  53 271 302 189 379  36 173 251  71
184 197 175
 108 382  15 352 348  61  84 217 191  98  11 202  69 315 215 377 190
322
 269 239 355 341 142 375  87 125 201 343 311 314 232  56  35 228 122
242
 206  57 291 324  25 237 156  52 252 371 353 216 310 360 279 192 327
210
 200 118 362 218 212 273 133 146  47  34 121  58 250 308  54 135  51
50
  19 214 178 231 101  18 226  62 120  88   6  79 372  44 112 154 317
91
 139 277 256 333 198 185 127 339  92  80  37  77   4  31 213  30 366
336
 261 179 386 329 342 280  95 188  68  13 307 113  93 361 132  43 193
124
 115 266 225 319 155 340 323 219 292 253 131 170  24 148 196 338 259
17
 276 205 110 176 387 378 109 262  67  45 358 186 152 325 243 144 157
281
 229  14 289 344 370 150  49 270 159 264  21 106  78 221 180  64  20
240
```

```
  26  23   5 312 241 141 299 103 128 283   0 158  76  29 227  55 364
74
 288  86   3 373 345 194 166 268 321 301 137 320 300 168 247 274 278
163
 224 182 174 107 265  41 290 297  38 129 368 296  96  81 234 161   1
33
 153 245 365  83 172 305 134 272 222  66 235 298  99 149 211  16 383
187
 316  46 171 208 326 357 248 335  85 376 294  22 136 164 104 384 332
9
 363 167 145 117 254 169 244 177 162 183  94 138 140 100 203 123 147
223
  10   8  40 160 119 105 328 287 385 195 346 267  48 102 249 381  60
233
 275 230 337 207 374  82 285  97 209 143  65 331 165 130 181 369 151
246
  90  63 282 116 111  12 304]
  Test:  [220 263  42 255 126 318 238 380 356 293  72 258 303  73 309
367 286 350
 199  27  28 313  70   2 257  59 330 349 284  32 204 306   7 347 351
354
 260 295  75]
```

Оцениваем качество оптимальной модели:

```
from sklearn.preprocessing import StandardScaler
X = cat_enc_oe[:, 0]
y = cat_enc_oe[:, 1]

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

X1 = X.reshape(-1, 1)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X1)
knn = KNeighborsClassifier(n_neighbors=5)
skf = StratifiedKFold(n_splits=3, shuffle=True, random_state=1)
cv_scores = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        X_scaled, y, scoring=scoring,
                        cv=3, return_train_score=True)

cv_scores

{'fit_time': array([0.0016253, 0.0010519, 0.0009973]),
 'score_time': array([0.01635671, 0.01194477, 0.00997305]),
 'test_precision': array([0.6775641 , 0.69149464, 0.70741403]),
 'train_precision': array([0.74211644, 0.70994257, 0.70186859]),
 'test_recall': array([0.75384615, 0.60465116, 0.63565891]),
 'train_recall': array([0.80232558, 0.64092664, 0.62548263]),
```

```
 'test_f1': array([0.71079165, 0.64075872, 0.66501973]),
 'train_f1': array([0.75602468, 0.6693309 , 0.65715174])}
```

Вывод: так как для обучающей выборки и тестовой результаты довольно близкие друг к другу, то можно сказать, что данная модель не недообучена и не переобучена. Параметр ближайших соседей, равный 5, является оптимальным.

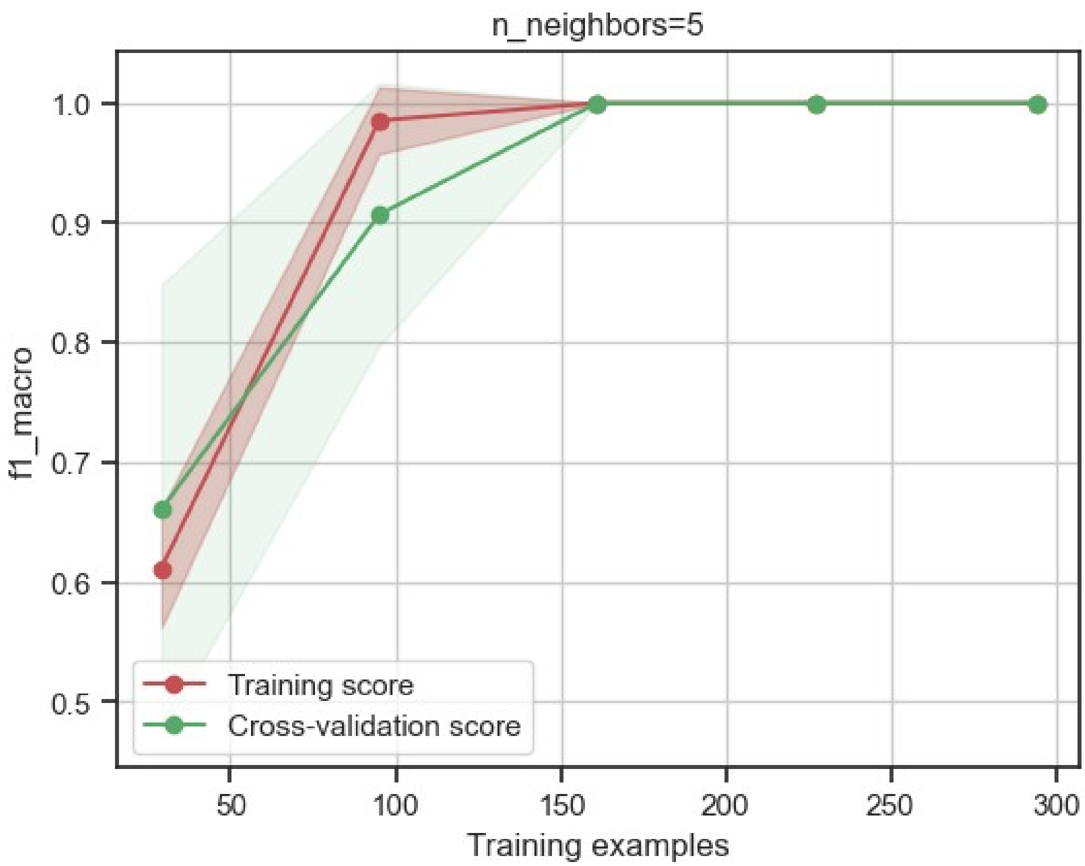
Построение кривых обучения и валидации

```
def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None,
                        n_jobs=None, train_sizes=np.linspace(.1, 1.0,
5), scoring='f1_weighted'):
    """
    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")
```

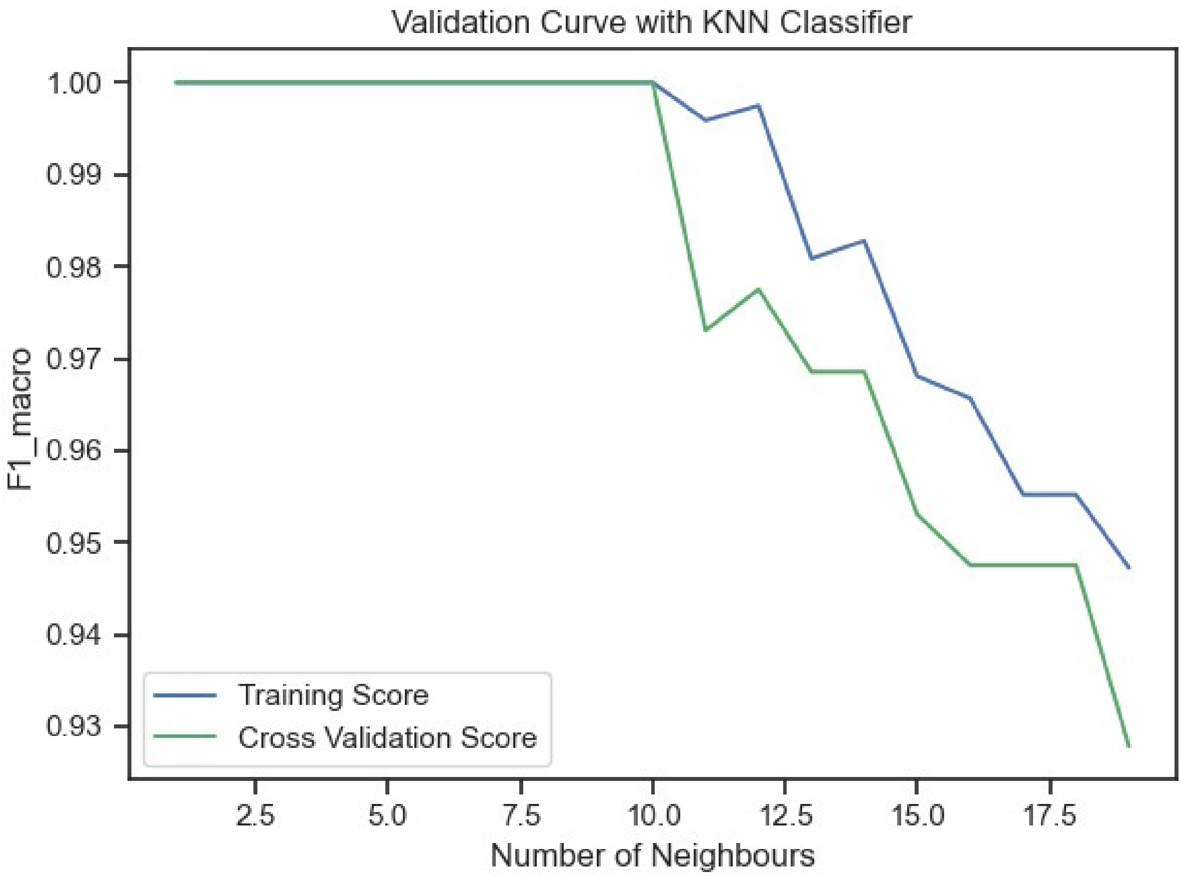
```
    plt.plot(train_sizes, train_scores_mean, 'o-', color="r",
             label="Training score")
    plt.plot(train_sizes, test_scores_mean, 'o-', color="g",
             label="Cross-validation score")

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

plot_learning_curve(KNeighborsClassifier(n_neighbors=5),
'n_neighbors=5',
                    data_x_train, data_y_train, cv=20,
scoring='f1_macro')
```

```
<module 'matplotlib.pyplot' from 'C:\\Users\\user\\anaconda3\\Lib\\
site-packages\\matplotlib\\pyplot.py'>
```




```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import load_digits
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import validation_curve
```

```python
data = np.c_[cat_enc_oe[:, 0], cat_enc_oe[:, 1]]
X, y = data, cat_enc_oe[:, 1]

# Define the range for the parameter (e.g., number of neighbors)
parameter_range = np.arange(1, 20, 1)

# Calculate accuracy on training and test set using the parameter with
cross-validation
train_score, test_score = validation_curve(
    KNeighborsClassifier(), X, y,
    param_name="n_neighbors",
    param_range=parameter_range,
    cv=15, scoring="f1_macro"
)

# Calculate mean and standard deviation of training and testing scores
mean_train_score = np.mean(train_score, axis=1)
std_train_score = np.std(train_score, axis=1)
mean_test_score = np.mean(test_score, axis=1)
std_test_score = np.std(test_score, axis=1)

# Plot mean accuracy scores for training and testing scores
plt.plot(parameter_range, mean_train_score, label="Training Score",
color='b')
plt.plot(parameter_range, mean_test_score, label="Cross Validation
Score", color='g')

# Create the plot
plt.title("Validation Curve with KNN Classifier")
plt.xlabel("Number of Neighbours")
plt.ylabel("F1_macro")
plt.tight_layout()
plt.legend(loc='best')
plt.show()
```



Validation Curve with KNN Classifier
1.00
0.99
0.98
0.97
0.96
0.95
0.94
0.93
F1_macro
Training Score
Cross Validation Score
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Number of Neighbours