Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Лабораторная работа №4
по дисциплине
«Методы машинного обучения»
на тему
«Подготовка обучающей и тестовой выборки, кросс-валидация и подбор гиперпараметров на примере метода ближайших соседей»

Выполнил:
студент группы ИУ5-22М
Бурашников В. В.

Москва — 2020 г.

1. Цель лабораторной работы

Изучить сложные способы подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей [?].

2. Задание

Требуется выполнить следующие действия [?]:

1. Выбрать набор данных (датасет) для решения задачи классификации или регресии.
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 трех подходящих для задачи метрик.
5. Постройте модель и оцените качество модели с использованием кросс-валидации. Проведите эксперименты с тремя различными стратегиями кросс-валидации.
6. Произведите подбор гиперпараметра K с использованием `GridSearchCV` и кросс-валидации.
7. Повторите пункт 4 для найденного оптимального значения гиперпараметра K. Сравните качество полученной модели с качеством модели, полученной в пункте 4.
8. Постройте кривые обучения и валидации.

3. Ход выполнения работы

Подключим все необходимые библиотеки и настроим отображение графиков [?, ?]:

In [2]:
```
from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, RepeatedKFold, ShuffleSplit
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.model_selection import learning_curve, validation_curve
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler

# Enable inline plots
%matplotlib inline

# Set plots formats to save high resolution PNG
from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")
```

Зададим ширину текстового представления данных, чтобы в дальнейшем текст в отчёте влезал на A4 [?]:

In [3]:
```
pd.set_option("display.width", 70)
```

3.1. Предварительная подготовка данных

В качестве набора данных используются оценки вина профессиональными сомелье и химические характеристики вин:

```
In [4]: data = pd.read_csv("./wine.csv")
```

Проверим полученные типы:

```
In [5]: data.dtypes
```

```
Out[5]: fixed acidity           float64
        volatile acidity        float64
        citric acid             float64
        residual sugar          float64
        chlorides               float64
        free sulfur dioxide     float64
        total sulfur dioxide    float64
        density                 float64
        pH                      float64
        sulphates               float64
        alcohol                 float64
        quality                   int64
        dtype: object
```

Посмотрим на данные в данном наборе данных:

```
In [6]: data.head()
```

```
Out[6]:    fixed acidity  volatile acidity  citric acid  residual sugar  \
        0            7.4              0.70         0.00             1.9
        1            7.8              0.88         0.00             2.6
        2            7.8              0.76         0.04             2.3
        3           11.2              0.28         0.56             1.9
        4            7.4              0.70         0.00             1.9

           chlorides  free sulfur dioxide  total sulfur dioxide  density  \
        0      0.076                 11.0                  34.0   0.9978
        1      0.098                 25.0                  67.0   0.9968
        2      0.092                 15.0                  54.0   0.9970
        3      0.075                 17.0                  60.0   0.9980
        4      0.076                 11.0                  34.0   0.9978

             pH  sulphates  alcohol  quality
        0  3.51       0.56      9.4        5
        1  3.20       0.68      9.8        5
        2  3.26       0.65      9.8        5
        3  3.16       0.58      9.8        6
        4  3.51       0.56      9.4        5
```

```
In [7]: df = data.copy()
        df.head()
```

```
Out[7]:    fixed acidity  volatile acidity  citric acid  residual sugar  \
        0            7.4              0.70         0.00             1.9
        1            7.8              0.88         0.00             2.6
        2            7.8              0.76         0.04             2.3
        3           11.2              0.28         0.56             1.9
        4            7.4              0.70         0.00             1.9

           chlorides  free sulfur dioxide  total sulfur dioxide  density  \
        0      0.076                 11.0                  34.0   0.9978
        1      0.098                 25.0                  67.0   0.9968
        2      0.092                 15.0                  54.0   0.9970
        3      0.075                 17.0                  60.0   0.9980
        4      0.076                 11.0                  34.0   0.9978

             pH  sulphates  alcohol  quality
        0  3.51       0.56      9.4        5
        1  3.20       0.68      9.8        5
        2  3.26       0.65      9.8        5
        3  3.16       0.58      9.8        6
        4  3.51       0.56      9.4        5
```

```
In [8]: df.dtypes
```

```
Out[8]: fixed acidity           float64
        volatile acidity        float64
        citric acid             float64
        residual sugar          float64
        chlorides               float64
        free sulfur dioxide     float64
        total sulfur dioxide    float64
        density                 float64
        pH                      float64
        sulphates               float64
        alcohol                 float64
        quality                   int64
        dtype: object
```

Проверим размер набора данных:

```
In [9]: df.shape
```

```
Out[9]: (1599, 12)
```

Проверим основные статистические характеристики набора данных:

```
In [10]: df.describe()
```

```
Out[10]:        fixed acidity  volatile acidity  citric acid  residual sugar  \
         count    1599.000000       1599.000000  1599.000000     1599.000000
         mean        8.319637          0.527821     0.270976        2.538806
         std         1.741096          0.179060     0.194801        1.409928
         min         4.600000          0.120000     0.000000        0.900000
```

```
25%         7.100000          0.390000     0.090000        1.900000
50%         7.900000          0.520000     0.260000        2.200000
75%         9.200000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide  \
count  1599.000000          1599.000000           1599.000000
mean      0.087467            15.874922             46.467792
std       0.047065            10.460157             32.895324
min       0.012000             1.000000              6.000000
25%       0.070000             7.000000             22.000000
50%       0.079000            14.000000             38.000000
75%       0.090000            21.000000             62.000000
max       0.611000            72.000000            289.000000

           density           pH    sulphates      alcohol  \
count  1599.000000  1599.000000  1599.000000  1599.000000
mean      0.996747     3.311113     0.658149    10.422983
std       0.001887     0.154386     0.169507     1.065668
min       0.990070     2.740000     0.330000     8.400000
25%       0.995600     3.210000     0.550000     9.500000
50%       0.996750     3.310000     0.620000    10.200000
75%       0.997835     3.400000     0.730000    11.100000
max       1.003690     4.010000     2.000000    14.900000

           quality
count  1599.000000
mean      5.636023
std       0.807569
min       3.000000
25%       5.000000
50%       6.000000
75%       6.000000
max       8.000000
```

Проверим наличие пропусков в данных:

```
In [11]: df.isnull().sum()

Out[11]: fixed acidity           0
         volatile acidity        0
         citric acid             0
         residual sugar          0
         chlorides               0
         free sulfur dioxide     0
         total sulfur dioxide    0
         density                 0
         pH                      0
         sulphates               0
         alcohol                 0
         quality                 0
         dtype: int64
```

3.2. Разделение данных

Разделим данные на целевой столбец и признаки:

```
In [67]: X = df.drop("density", axis=1)
         y = df["density"]

In [68]: print(X.head(), "\n")
         print(y.head())
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  \
0            7.4              0.70         0.00             1.9
1            7.8              0.88         0.00             2.6
2            7.8              0.76         0.04             2.3
3           11.2              0.28         0.56             1.9
4            7.4              0.70         0.00             1.9

   chlorides  free sulfur dioxide  total sulfur dioxide    pH  \
0      0.076                 11.0                  34.0  3.51
1      0.098                 25.0                  67.0  3.20
2      0.092                 15.0                  54.0  3.26
3      0.075                 17.0                  60.0  3.16
4      0.076                 11.0                  34.0  3.51

   sulphates  alcohol  quality
0       0.56      9.4        5
1       0.68      9.8        5
2       0.65      9.8        5
3       0.58      9.8        6
4       0.56      9.4        5

0    0.9978
1    0.9968
2    0.9970
3    0.9980
4    0.9978
Name: density, dtype: float64
```

```
In [69]: print(X.shape)
         print(y.shape)
```

```
(1599, 11)
(1599,)
```

Предобработаем данные, чтобы методы работали лучше:

```
In [70]: columns = X.columns
         scaler = StandardScaler()
         X = scaler.fit_transform(X)
         pd.DataFrame(X, columns=columns).describe()
```

```
/home/vladimir/PycharmProjects/giis_lab1/env/lib/python3.6/site-packages/sklearn/p
  return self.partial_fit(X, y)
/home/vladimir/PycharmProjects/giis_lab1/env/lib/python3.6/site-packages/sklearn/b
  return self.fit(X, **fit_params).transform(X)
```

Out[70]:

	fixed acidity	volatile acidity	citric acid
count	1.599000e+03	1.599000e+03	1.599000e+03
mean	3.554936e-16	1.733031e-16	-8.887339e-17
std	1.000313e+00	1.000313e+00	1.000313e+00
min	-2.137045e+00	-2.278280e+00	-1.391472e+00
25%	-7.007187e-01	-7.699311e-01	-9.293181e-01
50%	-2.410944e-01	-4.368911e-02	-5.636026e-02
75%	5.057952e-01	6.266881e-01	7.652471e-01
max	4.355149e+00	5.877976e+00	3.743574e+00

	residual sugar	chlorides	free sulfur dioxide
count	1.599000e+03	1.599000e+03	1.599000e+03
mean	-1.244227e-16	3.821556e-16	-6.221137e-17
std	1.000313e+00	1.000313e+00	1.000313e+00
min	-1.162696e+00	-1.603945e+00	-1.422500e+00
25%	-4.532184e-01	-3.712290e-01	-8.487156e-01
50%	-2.403750e-01	-1.799455e-01	-1.793002e-01
75%	4.341614e-02	5.384542e-02	4.901152e-01
max	9.195681e+00	1.112703e+01	5.367284e+00

	total sulfur dioxide	pH	sulphates
count	1.599000e+03	1.599000e+03	1.599000e+03
mean	4.443669e-17	2.861723e-15	6.754377e-16
std	1.000313e+00	1.000313e+00	1.000313e+00
min	-1.230584e+00	-3.700401e+00	-1.936507e+00
25%	-7.440403e-01	-6.551405e-01	-6.382196e-01
50%	-2.574968e-01	-7.212705e-03	-2.251281e-01
75%	4.723184e-01	5.759223e-01	4.240158e-01
max	7.375154e+00	4.528282e+00	7.918677e+00

	alcohol	quality
count	1.599000e+03	1.599000e+03
mean	1.066481e-16	8.887339e-17
std	1.000313e+00	1.000313e+00
min	-1.898919e+00	-3.265165e+00
25%	-8.663789e-01	-7.878226e-01
50%	-2.093081e-01	4.508484e-01
75%	6.354971e-01	4.508484e-01
max	4.202453e+00	2.928190e+00

Разделим выборку на тренировочную и тестовую:

```
In [71]: X_train, X_test, y_train, y_test = train_test_split(X, y,
                                        test_size=0.25, random_state=346705925)
```

```
In [72]: print(X_train.shape)
         print(X_test.shape)
         print(y_train.shape)
         print(y_test.shape)

(1199, 11)
(400, 11)
(1199,)
(400,)
```

3.3. Модель ближайших соседей для произвольно заданного гиперпараметра K

Напишем функцию, которая считает метрики построенной модели:

```
In [73]: def test_model(model):
             print("mean_absolute_error:",
                   mean_absolute_error(y_test, model.predict(X_test)))
             print("median_absolute_error:",
                   median_absolute_error(y_test, model.predict(X_test)))
             print("r2_score:",
                   r2_score(y_test, model.predict(X_test)))
```

Попробуем метод ближайших соседей с гиперпараметром $K = 5$:

```
In [74]: reg_5 = KNeighborsRegressor(n_neighbors=5)
         reg_5.fit(X_train, y_train)

Out[74]: KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                   metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                   weights='uniform')
```

Проверим метрики построенной модели:

```
In [75]: test_model(reg_5)

mean_absolute_error: 0.0006631799999999915
median_absolute_error: 0.0004889999999999617
r2_score: 0.7565853831960054
```

Видно, что средние ошибки не очень показательны для одной модели, они больше подходят для сравнения разных моделей. В тоже время коэффициент детерминации неплох сам по себе, в данном случае модель более-менее состоятельна.

3.4. Использование кросс-валидации

Проверим различные стратегии кросс-валидации. Для начала посмотрим классический K-fold:

```
In [76]: scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                                  cv=KFold(n_splits=10), scoring="r2")
         print(scores)
         print(scores.mean(), "±", scores.std())
```

```
[0.54201038 0.62530238 0.60103483 0.42462355 0.16800613 0.67138819
 0.69157226 0.54801332 0.58253055 0.60540914]
0.5459890743612823 ± 0.14447376628869613
```

```
In [77]: scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                                  cv=RepeatedKFold(n_splits=5, n_repeats=2),
                                  scoring="r2")
         print(scores)
         print(scores.mean(), "±", scores.std())
```

```
[0.77705867 0.75730479 0.73613982 0.73003897 0.75598591 0.75393584
 0.77560511 0.76695948 0.73507993 0.70824869]
0.7496357209085482 ± 0.020820903031886513
```

```
In [78]: scores = cross_val_score(KNeighborsRegressor(n_neighbors=5), X, y,
                                  cv=ShuffleSplit(n_splits=10), scoring="r2")
         print(scores)
         print(scores.mean(), "±", scores.std())
```

```
[0.80127744 0.78002178 0.77815453 0.70739327 0.77181739 0.75596438
 0.77531645 0.70537579 0.75995107 0.74154166]
0.7576813769129686 ± 0.029770859778064405
```

3.5. Подбор гиперпараметра K

Введем список настраиваемых параметров:

```
In [79]: n_range = np.array(range(1, 50, 2))
         tuned_parameters = [{'n_neighbors': n_range}]
         n_range
```

```
Out[79]: array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33
                35, 37, 39, 41, 43, 45, 47, 49])
```

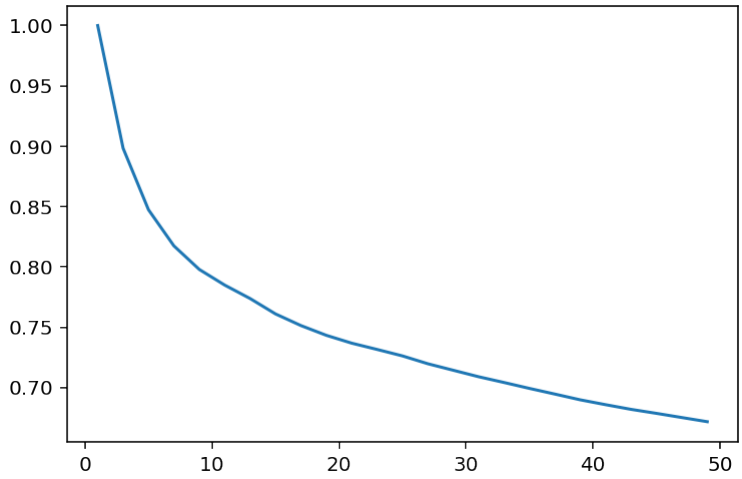
Запустим подбор параметра:

```
In [80]: gs = GridSearchCV(KNeighborsRegressor(), tuned_parameters,
                           cv=ShuffleSplit(n_splits=10), scoring="r2",
                           return_train_score=True, n_jobs=-1)
         gs.fit(X, y)
         gs.best_params_
```

```
Out[80]: {'n_neighbors': 3}
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе данных:

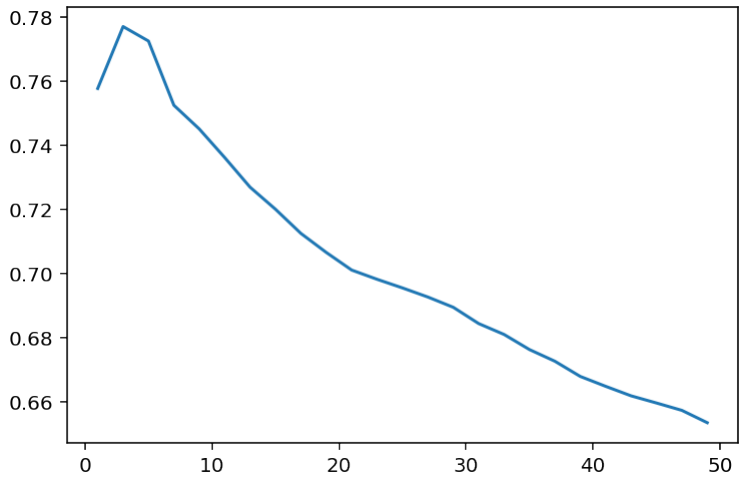
```
In [81]: plt.plot(n_range, gs.cv_results_["mean_train_score"]);
```



Очевидно, что для $K = 1$ на тренировочном наборе данных мы находим ровно ту же точку, что и нужно предсказать, и чем больше её соседей мы берём — тем меньше точность.

На тестовом наборе данных картина сильно интереснее:

```
In [82]: plt.plot(n_range, gs.cv_results_["mean_test_score"]);
```



Выходит, что сначала соседей слишком мало (высоко влияние выбросов), а затем количество соседей постепенно становится слишком велико, и среднее значение по этим соседям всё больше и больше оттягивает значение от истинного.

Проверим получившуюся модель:

```
In [83]: reg = KNeighborsRegressor(**gs.best_params_)
         reg.fit(X_train, y_train)
         test_model(reg)

mean_absolute_error: 0.0006740416666666596
median_absolute_error: 0.0005033333333333001
r2_score: 0.7550997432386466
```

В целом получили примерно тот же результат. Очевидно, что проблема в том, что данный метод и так показал достаточно хороший результат для данной выборки.

Построим кривую обучения [?]:

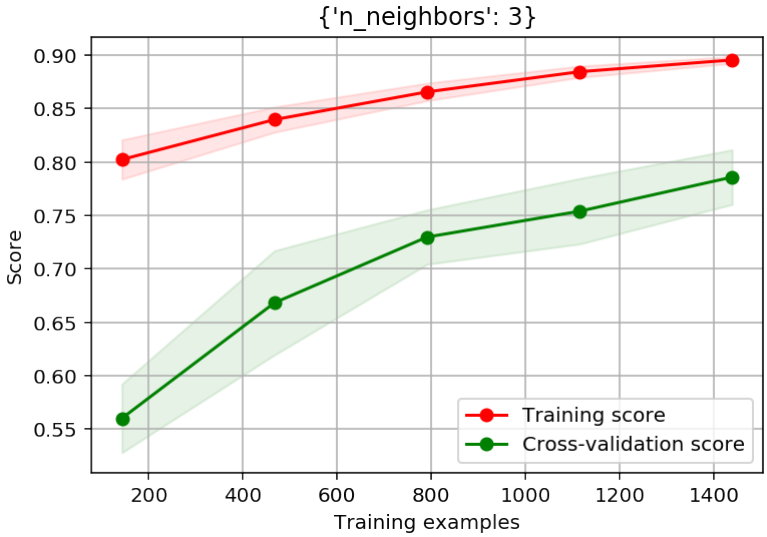
```
In [84]: def plot_learning_curve(estimator, title, X, y, ylim=None, cv=None):
             train_sizes=np.linspace(.1, 1.0, 5)

             plt.figure()
             plt.title(title)
             if ylim is not None:
                 plt.ylim(*ylim)
             plt.xlabel("Training examples")
             plt.ylabel("Score")
             train_sizes, train_scores, test_scores = learning_curve(
                 estimator, X, y, cv=cv, n_jobs=-1, 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=0.1,
                              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 [85]: plot_learning_curve(reg, str(gs.best_params_), X, y,
                             cv=ShuffleSplit(n_splits=10));
```



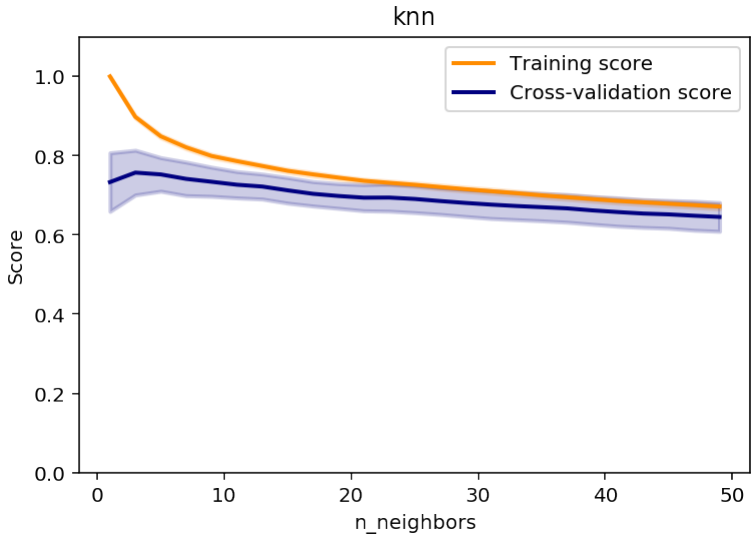
Построим кривую валидации:

```
In [86]: 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("Score")
             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=0.2,
                              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 [87]:
```
plot_validation_curve(KNeighborsRegressor(), "knn", X, y,
                      param_name="n_neighbors", param_range=n_range,
                      cv=ShuffleSplit(n_splits=10), scoring="r2");
```



In [ ]: