

Московский государственный технический университет им. Н.Э. Баумана  
Факультет «Информатика и системы управления»  
Кафедра «Автоматизированные системы обработки информации и  
управления»



**Отчет**  
**Лабораторная работа № 3**  
**По курсу «Технологии машинного обучения»**

**ИСПОЛНИТЕЛЬ:**

Группа ИУ5-65Б

Камалов М.Р.

"23" мая 2021 г.

**ПРЕПОДАВАТЕЛЬ:**

Гапанюк Ю.Е.

\_\_\_\_\_

"\_\_" \_\_\_\_\_ 2021 г.

Москва 2021

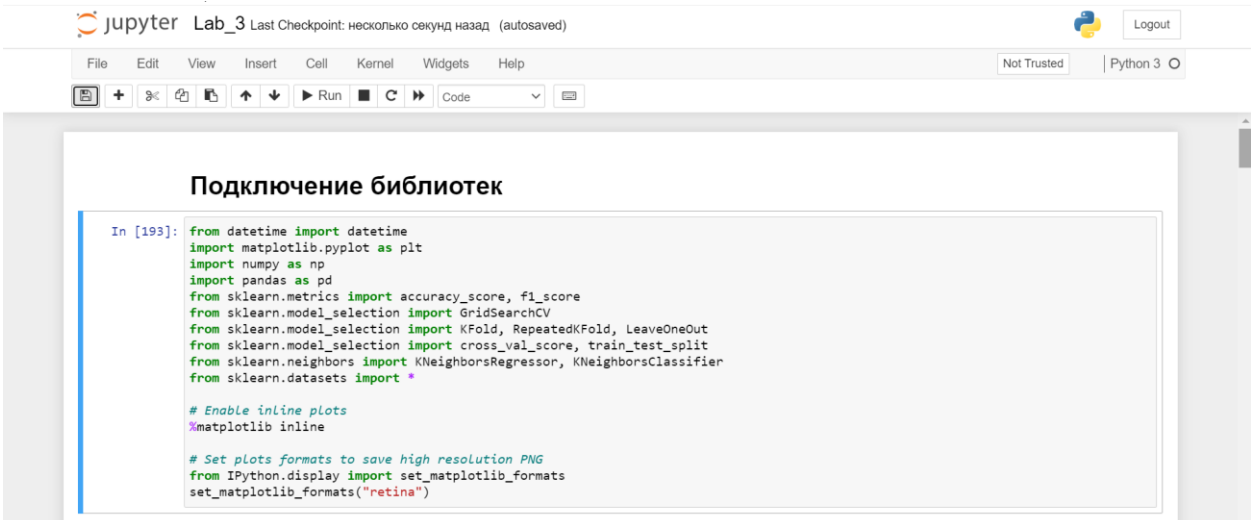
---

**1. Цель лабораторной работы:** изучение способов подготовки выборки и подбора гиперпараметров на примере метода ближайших соседей.

## 2. 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.

## 3. Скриншоты jupyter notebook



```
In [193]: from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.metrics import accuracy_score, f1_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold, RepeatedKFold, LeaveOneOut
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.datasets import import *

# Enable inline plots
%matplotlib inline

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

## Загружаем датасет

```
In [36]: wine = load_wine()
```

```
In [21]: # Наименование признаков
         wine.feature_names
```

```
Out[21]: ['alcohol',
          'malic_acid',
          'ash',
          'alcalinity_of_ash',
          'magnesium',
          'total_phenols',
          'flavanoids',
          'nonflavanoid_phenols',
          'proanthocyanins',
          'color_intensity',
          'hue',
          'od280/od315_of_diluted_wines',
          'proline']
```

```
In [23]: # Значения целевого признака
np.unique(wine.target)
```

```
Out[23]: array([0, 1, 2])
```

```
In [26]: # Наименования значений целевого признака
         wine.target_names
```

```
Out[26]: array(['class_0', 'class_1', 'class_2'], dtype='<U7')
```

```
In [27]: list(zip(np.unique(wine.target), wine.target_names))
```

```
Out[27]: [(0, 'class_0'), (1, 'class_1'), (2, 'class_2')]
```

```
In [28]: # Значения целевого признака
         wine.target
```

[illegible]

```
In [29]: # Размер выборки
         wine.data.shape, wine.target.shape
```

```
Out[29]: ((178, 13), (178,))
```

```
In [15]: def make_dataframe(ds_function):
          ds = ds_function()
          df = pd.DataFrame(data= np.c_[ds['data'], ds['target']],
                           columns= list(ds['feature_names']) + ['target'])
          return df
```

```
In [32]: # Сформируем DataFrame
         wine_df = make_dataframe(load_wine)
         wine_df.head()
```

|   | alcohol | malic_acid | ash  | alcalinity_of_ash | magnesium | total_phenols | flavanoids | nonflavanoid_phenols | proanthocyanins | color_intensity | hue  | od280/od31 |
|---|---------|------------|------|-------------------|-----------|---------------|------------|----------------------|-----------------|-----------------|------|------------|
| 0 | 14.23   | 1.71       | 2.43 |                   | 15.6      | 127.0         | 2.80       | 3.06                 | 0.28            | 2.29            | 5.64 | 1.04       |
| 1 | 13.20   | 1.78       | 2.14 |                   | 11.2      | 100.0         | 2.65       | 2.76                 | 0.26            | 1.28            | 4.38 | 1.05       |
| 2 | 13.16   | 2.36       | 2.67 |                   | 18.6      | 101.0         | 2.80       | 3.24                 | 0.30            | 2.81            | 5.68 | 1.03       |
| 3 | 14.37   | 1.95       | 2.50 |                   | 16.8      | 113.0         | 3.85       | 3.49                 | 0.24            | 2.18            | 7.80 | 0.86       |
| 4 | 13.24   | 2.59       | 2.87 |                   | 21.0      | 118.0         | 2.80       | 2.69                 | 0.39            | 1.82            | 4.32 | 1.04       |

```
In [34]: # Выведем его статистические характеристики
         wine_df.describe()
```

```
Out[34]:
```

|       | 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       |

```
Out[35]: alcohol      0
malic_acid           0
ash                  0
alcalinity_of_ash    0
magnesium            0
total_phenols        0
flavanoids           0
nonflavanoid_phenols 0
proanthocyanins      0
color_intensity      0
hue                  0
od280/od315_of_diluted_wines 0
proline              0
target               0
dtype: int64
```

```
In [179]: X, y = load_wine( return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split( X, y, test_size = 0.15, random state = 1 )
```

```
In [180]: # Размер обучающей и тестовой выборки
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

(151, 13)  
(27, 13)  
(151, )  
(27, )

```
In [181]: cl1_1 = KNeighborsClassifier(n_neighbors=50)
          cl1_1.fit(X_train, y_train)
          target1_0 = cl1_1.predict(X_train)
          target1_1 = cl1_1.predict(X_test)
          accuracy_score(y_train, target1_0), accuracy_score(y_test, target1_1)
```

```
Out[181]: (0.7350993377483444, 0.6666666666666666)
```

```
In [156]: # Параметры TP, TN, FP, FN считаются как сумма по всем классам
f1_score(y_test, target1_1, average='micro')
```

Out[156]: 0.7083333333333333

```
In [157]: # Параметры TP, TN, FP, FN считаются отдельно для каждого класса
# и берется среднее значение, дисбаланс классов не учитывается.
f1 score(y_test, target1_1, average='macro')
```

Out[157]: 0.6716285663654085

```
In [104]: # 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).
f1_score(y_test, target1, average='weighted')
```

Out[104]: 0.6323169433801618

```
In [135]: # LeaveOneOut стратегия (в тестовую выборку помещается всего один элемент)
scores = cross_val_score(KNeighborsClassifier(n_neighbors=50),
                          X, y, cv=LeaveOneOut())
scores, np.mean(scores)
```

```
Out[135]: (array([[1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,  
1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1., 1.,  
1., 1., 1., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1.,  
1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0., 1., 1.,  
0., 0., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 0., 0., 1., 1.,  
1., 1., 0., 0., 0., 1., 1., 1., 1., 1., 0., 0., 1., 0., 1., 0., 0.,  
0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 1.,  
1., 0., 1., 1., 1., 1., 0., 0., 0., 1., 0., 1., 1., 0., 0., 0.,  
1., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 1.,  
0., 0., 1., 1., 1., 1., 1., 0., 1.]],
```

[illegible]

```
Out[137]: (array([0.72222222, 0.66666667, 0.66666667, 0.77142857, 0.8
0.7253968253968253]))
```

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

```
In [118]: n_range = np.array(range(2,100,2))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

```
Out[118]: [{'n_neighbors': array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34,
        36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68,
        70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98])}]
```

```
In [187]: %%time
gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=7, scoring='accuracy')
gs.fit(X, y)

Wall time: 1.21 s
```

```
Out[187]: GridSearchCV(cv=7, estimator=KNeighborsClassifier(),
        param_grid=[{'n_neighbors': array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30, 32, 34,
        36, 38, 40, 42, 44, 46, 48, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68,
        70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90, 92, 94, 96, 98])}],
        scoring='accuracy')
```

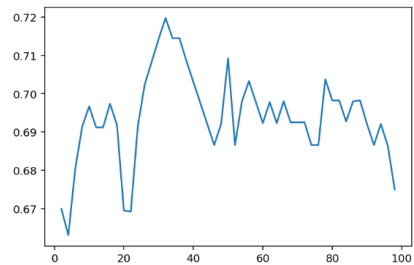
```
In [188]: gs.best_estimator_
```

```
Out[188]: KNeighborsClassifier(n_neighbors=32)
```

```
In [189]: gs.best_params_
```

```
Out[189]: {'n_neighbors': 32}
```

```
In [190]: plt.plot(n_range, gs.cv_results_["mean_test_score"]);
```



## Обучение модели и оценка качества с учетом подобранных гиперпараметров

```
In [191]: gs.best_estimator_.fit(X_train, y_train)
target2_0 = gs.best_estimator_.predict(X_train)
target2_1 = gs.best_estimator_.predict(X_test)
```

```
In [192]: # Новое качество модели
accuracy_score(y_train, target2_0), accuracy_score(y_test, target2_1)
```

```
Out[192]: (0.7417218543046358, 0.7037037037037037)
```

```
In [182]: # Качество модели до подбора гиперпараметров
accuracy_score(y_train, target1_0), accuracy_score(y_test, target1_1)
```

```
Out[182]: (0.7350993377483444, 0.6666666666666666)
```