

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
им. Н.Э. Баумана

---

Факультет «Информатика и система управления»  
Кафедра ИУ-5 «Системы обработки информации и управления»

ОТЧЕТ

Лабораторная работа №4 по  
курсу «Методы машинного обучения»

Исполнитель - студент группы ИУ5-21М:

Кауров Максим \_\_\_\_\_

Москва – 2020 год

```

from google.colab import files

import numpy as np
import pandas as pd
import seaborn as sns
import sklearn.impute
import sklearn.preprocessing
import matplotlib.pyplot as plt


%matplotlib inline

sns.set(style="ticks")

from IPython.display import set_matplotlib_formats
set_matplotlib_formats("retina")

uploaded = files.upload()

for fn in uploaded.keys():
    print('User uploaded file "{name}" with length {length} bytes'.format(
        name=fn, length=len(uploaded[fn])))

 Upload widget Файл не выбран available when the cell has been execute
the current browser session. Please rerun this cell to enable.
Saving PatientInfo.csv to PatientInfo (2).csv
User uploaded file "PatientInfo.csv" with length 220884 bytes


data = pd.read_csv('PatientInfo.csv')

```

## ▼ Информация о датасете

В качестве датасета для лабораторной работы был выбран датасет с подтвержденной инф COVID-19 в Южной Кореи

```
data.head()
```

 **Saving...**

	patient_id	global_num	sex	birth_year	age	country	province	city	dise
0	1000000001	2.0	male	1964.0	50s	Korea	Seoul	Gangseo-gu	
1	1000000002	5.0	male	1987.0	30s	Korea	Seoul	Junngnang-gu	
2	1000000003	6.0	male	1964.0	50s	Korea	Seoul	Jongno-gu	
3	1000000004	7.0	male	1991.0	20s	Korea	Seoul	Mapo-gu	
4	1000000005	9.0	female	1992.0	20s	Korea	Seoul	Seongbuk-gu	

# Цель лабораторной работы

Изучить сложные способы подготовки выборки и подбора гиперпараметров на примере м

## Задание

Требуется выполнить следующие действия:

1. Выбрать набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирован
3. С использованием метода `train_test_split` разделите выборку на обучающую и тест
4. Обучите модель ближайших соседей для произвольно заданного гиперпараметра  $K$ . трех подходящих для задачи метрик.
5. Постройте модель и оцените качество модели с использованием кросс-валидации. П различными стратегиями кросс-валидации.
6. Произведите подбор гиперпараметра  $K$  с использованием `GridSearchCV` и кросс-вал
7. Повторите пункт 4 для найденного оптимального значения гиперпараметра  $K$ . Срав с качеством модели, полученной в пункте 4.
8. Постройте кривые обучения и валидации.

## ▼ Ход выполнения работы

Подключим все необходимые библиотеки и настроим отображение графиков:

```
from google.colab import files
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 в.

```
pd.set_option("display.width", 70)
```

## ▼ Предварительная подготовка данных

В качестве набора данных используем датасет со статистикой игр в TFT

```
uploaded = files.upload()
```

Выбрать файлы TFT\_Plati...hData.csv

- **TFT\_Platinum\_MatchData.csv**(application/vnd.ms-excel) - 40812862 bytes, last modified: 24.05.2020  
Saving TFT\_Platinum\_MatchData.csv to TFT\_Platinum\_MatchData.csv

```
data = pd.read_csv("TFT_Platinum_MatchData.csv")
```

Проверим полученные типы:

```
data.dtypes
```

```
gameId          object
gameDuration    float64
level           int64
lastRound       int64
Ranked          int64
ingameDuration  float64
combination     object
champion        object
dtype: object
```

Посмотрим на данные в данном наборе данных:

```
data.head()
```

```
gameDuration  level  lastRound  Ranked  ingameDuration
0    1963.905273     6         27      5    1390.165771
1    1963.905273     8         37      3    1891.282715
2    1963.905273     6         25      7    1279.461060
3    1963.905273     7         38      2    1955.608521
4    1963.905273     8         38      1    1955.608521
```

Избавимся от не числовых данных

```
data.drop(['gameId', 'combination', 'champion'],axis='columns', inplace=True)
```

Проверим размер набора данных:

```
data.shape
```

```
↳ (80000, 5)
```

Проверим основные статистические характеристики набора данных:

```
data.describe()
```

```
↳
```

	gameDuration	level	lastRound	Ranked	ingameDuration
<b>count</b>	80000.000000	80000.000000	80000.000000	80000.000000	80000.000000
<b>mean</b>	2168.427079	7.785637	33.193112	4.496375	1881.922944
<b>std</b>	156.294695	0.828188	4.878634	2.294254	279.213257
<b>min</b>	0.000000	1.000000	0.000000	0.000000	0.000000
<b>25%</b>	2073.243225	7.000000	30.000000	2.000000	1705.082611
<b>50%</b>	2164.299072	8.000000	34.000000	4.000000	1903.187744
<b>75%</b>	2261.792358	8.000000	37.000000	6.000000	2076.401306
<b>max</b>	2714.283691	9.000000	48.000000	8.000000	2706.018555

Проверим наличие пропусков в данных:

```
data.isnull().sum()
```

```
↳ gameDuration    0
   level          0
   lastRound      0
   Ranked         0
   ingameDuration  0
   dtype: int64
```

## ▼ Разделение данных

Разделим данные на целевой столбец и признаки:

```
y = data["level"]
X = data.drop("level", axis=1)
```

```
print(X.head(), "\n")
print(y.head())
```

```
↳
```

	gameDuration	lastRound	Ranked	ingameDuration
0	1963.905273	27	5	1390.165771
1	1963.905273	37	3	1891.282715
2	1963.905273	25	7	1279.461060
3	1963.905273	38	2	1955.608521
4	1963.905273	38	1	1955.608521

0	6
1	8
2	6
3	7
4	8

```
print(X.shape)
print(y.shape)
```

```
↳ (80000, 4)
   (80000,)
```

Предобработаем данные, чтобы методы работали лучше:

```
columns = X.columns
scaler = StandardScaler()
X = scaler.fit_transform(X)
pd.DataFrame(X, columns=columns).describe()
```

```
↳
```

	gameDuration	lastRound	Ranked	ingameDuration
<b>count</b>	8.000000e+04	8.000000e+04	8.000000e+04	8.000000e+04
<b>mean</b>	-8.050338e-16	-1.837815e-16	1.831951e-16	-1.173825e-16
<b>std</b>	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00
<b>min</b>	-1.387405e+01	-6.803815e+00	-1.959854e+00	-6.740134e+00
<b>25%</b>	-6.090063e-01	-6.545137e-01	-1.088105e+00	-6.333562e-01
<b>50%</b>	-2.641185e-02	1.653931e-01	-2.163571e-01	7.616019e-02
<b>75%</b>	5.973706e-01	7.803232e-01	6.553913e-01	6.965270e-01
<b>max</b>	3.492505e+00	3.035067e+00	1.527140e+00	2.951510e+00

Разделим выборку на тренировочную и тестовую:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.25, random_state=346705925)
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
↳
```

```
(60000, 4)
(20000, 4)
```

## ▼ Модель ближайших соседей для произвольно заданного гипер

Напишем функцию, которая считает метрики построенной модели:

```
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  $K = 4$ :

```
reg_4 = KNeighborsRegressor(n_neighbors=4)
reg_4.fit(X_train, y_train)

☞ KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
                       metric_params=None, n_jobs=None, n_neighbors=4, p=2,
                       weights='uniform')
```

Проверим метрики построенной модели:

```
test_model(reg_4)

☞ mean_absolute_error: 0.5681125
   median_absolute_error: 0.5
   r2_score: 0.2421847409099145
```

Видно, что средние ошибки не очень показательны для одной модели, они больше подходят для сравнения. Коэффициент детерминации неплох сам по себе, в данном случае модель более-менее

## ▼ Использование кросс-валидации

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

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

☞ [0.2467813  0.3368431  0.32030886 0.24195008 0.22140974 0.21119448
    0.25057293 0.40447677 0.26440955 0.22364643]
    0.27215932344368743 ± 0.05893953047717207
```

```
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())
```

```
↳ [0.27631682 0.27782999 0.28106601 0.28287685 0.26800889 0.28644
    0.28255793 0.26524816 0.2662985 0.28312384]
    0.276976699710253 ± 0.0073701666317066705
```

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

```
↳ [0.2772828 0.31235754 0.27398576 0.31175471 0.30951986 0.28455921
    0.29369173 0.29589917 0.22481847 0.23117086]
    0.2815040105466086 ± 0.029728852968591082
```

## ▼ Подбор гиперпараметра $K$

Введем список настраиваемых параметров:

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

```
↳ 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:

```
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_
```

```
↳ {'n_neighbors': 49}
```

Проверим результаты при разных значения гиперпараметра на тренировочном наборе да

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

```
↳
```



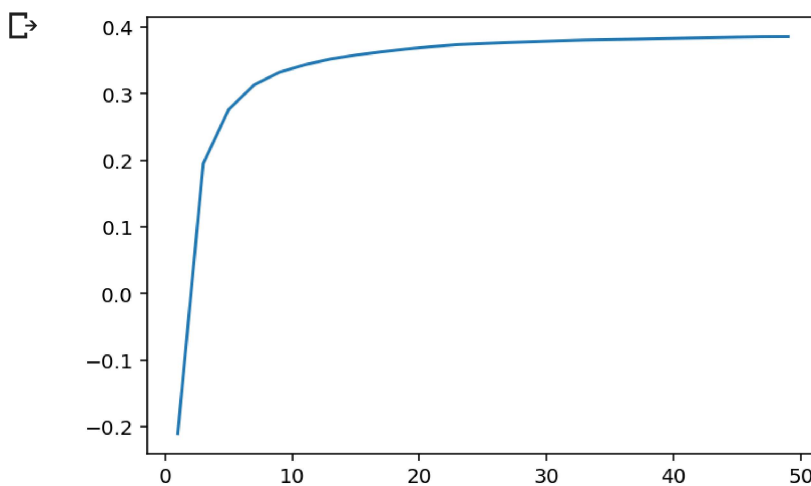


Очевидно, что для  $K = 1$  на тренировочном наборе данных мы находим ровно ту же точку её соседей мы берём — тем меньше точность.



Картинка с тестового набора данных:

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



Судя по полученным результатам, выбранный датасет не слишком подходит для данного :

Проверим получившуюся модель:

```
reg = KNeighborsRegressor(**gs.best_params_)
reg.fit(X_train, y_train)
test_model(reg)
```

```
mean_absolute_error: 0.5172234693877551
median_absolute_error: 0.4285714285714288
r2_score: 0.3824127307424263
```

В целом получили примерно тот же результат. Очевидно, что проблема в том, что данный результат для данной выборки.

Построим кривую обучения:

```
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")

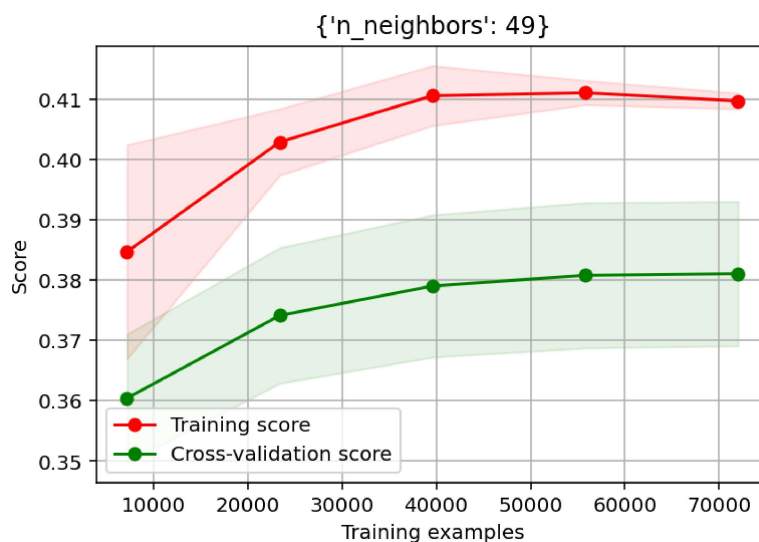
plt.legend(loc="best")
return plt

```

```

plot_learning_curve(reg, str(gs.best_params_), X, y,
                    cv=ShuffleSplit(n_splits=10));

```



Построим кривую валидации:

```

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

```

```

plot_validation_curve(KNeighborsRegressor(), "knn", X, y,
                     param_name="n_neighbors", param_range=n_range,
                     cv=ShuffleSplit(n_splits=10), scoring="r2");

```

