



**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 \_\_\_\_\_

Отчет по лабораторной работе №6  
**«Ансамбли моделей машинного обучения»**  
по курсу «Технологии машинного обучения»

Выполнил:  
Студент группы ИУ5Ц-81Б  
Тураев Глеб

\_\_\_\_\_

Проверил:  
Преподаватель кафедры ИУ5  
Гапанюк Ю.Е.

\_\_\_\_\_

---

**Москва 2020**

---

**Цель лабораторной работы:** изучение ансамблей моделей машинного обучения.

**Задание:**

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 две ансамблевые модели. Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.

**Выполнение лабораторной работы:**

**1. Импортируем библиотеки:**

Осуществим импорт библиотек с помощью команды **import**:

```
[29] from datetime import datetime
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import accuracy_score, balanced_accuracy_score

%matplotlib inline

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

Для ЛР№6 я использовал набор данных "Admission\_Predict"(параметры, которые важны при поступлении на магистерские программы, а также позволяющие поступить выпускникам в те или иные ВУЗы), так как считаю, что методы машинного обучения можно использовать на данных студентов, тем более мы все сами студенты и данный набор для нас ближе.

```
[30] data = pd.read_csv("/Admission_Predict.csv")
```

```
[31] data.dtypes
```

```
Serial No.      int64
GRE Score       int64
TOEFL Score     int64
University Rating int64
SOP            float64
LOR            float64
CGPA           float64
Research       int64
Chance of Admit float64
dtype: object
```

```
[32] data.head()
```

```
Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research  Chance of Admit
0           1         337         118              4  4.5  4.5  9.65         1         0.92
1           2         324         107              4  4.0  4.5  8.87         1         0.76
2           3         316         104              3  3.0  3.5  8.00         1         0.72
3           4         322         110              3  3.5  2.5  8.67         1         0.80
4           5         314         103              2  2.0  3.0  8.21         0         0.65
```

```
[33] df = data.copy()
```

## Узнаем размер данного набора данных:

```
[34] df.shape
```

```
(400, 9)
```

## Статистические характеристики набора данных:

```
[35] df.describe()
```

```
Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Research  Chance of Admit
count  400.000000  400.000000  400.000000  400.000000  400.000000  400.000000  400.000000  400.000000  400.000000
mean    200.500000  316.807500  107.410000    3.087500    3.400000    3.452500    8.598925    0.547500    0.724350
std     115.614301  11.473646    6.069514    1.143728    1.006869    0.898478    0.596317    0.498362    0.142609
min      1.000000  290.000000   92.000000    1.000000    1.000000    1.000000    6.800000    0.000000    0.340000
25%    100.750000  308.000000  103.000000    2.000000    2.500000    3.000000    8.170000    0.000000    0.640000
50%    200.500000  317.000000  107.000000    3.000000    3.500000    3.500000    8.610000    1.000000    0.730000
75%    300.250000  325.000000  112.000000    4.000000    4.000000    4.000000    9.062500    1.000000    0.830000
max     400.000000  340.000000  120.000000    5.000000    5.000000    5.000000    9.920000    1.000000    0.970000
```

## 2. Наличие пропусков в данных:

```
[36] df.isnull().sum()
```

```
Serial No.      0
GRE Score       0
TOEFL Score     0
University Rating 0
SOP             0
LOR             0
CGPA            0
Research        0
Chance of Admit  0
dtype: int64
```

### 3. Разделение данных:

С использованием метода `train_test_split` разделите выборку на обучающую и тестовую.

Для начала разделим данные на целевой столбец и признаки:

```
[37] X = df.drop("Research", axis = 1)
      y = df["Research"]
```

Результат:

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

```
Serial No.  GRE Score  TOEFL Score  ...  LOR  CGPA  Chance of Admit
0           1        337           118  ...   4.5   9.65             0.92
1           2        324           107  ...   4.5   8.87             0.76
2           3        316           104  ...   3.5   8.00             0.72
3           4        322           110  ...   2.5   8.67             0.80
4           5        314           103  ...   3.0   8.21             0.65
```

[5 rows x 8 columns]

```
0    1
1    1
2    1
3    1
4    0
Name: Research, dtype: int64
```

```
[39] print(X.shape)
      print(y.shape)
```

```
(400, 8)
(400,)
```

Предобработка данных:

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

```
Serial No.  GRE Score  TOEFL Score  University Rating  SOP  LOR  CGPA  Chance of Admit
count  4.000000e+02  4.000000e+02  4.000000e+02  4.000000e+02  4.000000e+02  4.000000e+02  4.000000e+02  4.000000e+02
mean    6.383782e-17  -3.785861e-16  5.412337e-16  7.147061e-16  1.859624e-16  -3.019807e-16  8.076873e-16  -3.314016e-16
std     1.001252e+00  1.001252e+00  1.001252e+00  1.001252e+00  1.001252e+00  1.001252e+00  1.001252e+00  1.001252e+00
min    -1.727726e+00  -2.339367e+00  -2.542098e+00  -1.827457e+00  -2.386613e+00  -2.733036e+00  -3.020504e+00  -2.698500e+00
25%    -8.638630e-01  -7.685900e-01  -7.274920e-01  -9.520286e-01  -8.949798e-01  -5.042604e-01  -7.201909e-01  -5.922168e-01
50%     0.000000e+00  1.679859e-02  -6.763531e-02  -7.660001e-02  9.944220e-02  5.293342e-02  1.859559e-02  3.966834e-02
75%     8.638630e-01  7.149218e-01  7.571856e-01  7.988286e-01  5.966532e-01  6.101273e-01  7.783704e-01  7.417629e-01
max     1.727726e+00  2.023903e+00  2.076899e+00  1.674257e+00  1.591075e+00  1.724515e+00  2.218165e+00  1.724695e+00
```

## Осуществим разделение выборки на тренировочную и тестовую:

```
[41] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state = 387235673)
```

```
[42] print(X_train.shape)
      print(X_test.shape)
      print(y_train.shape)
      print(y_test.shape)
```

```
↳ (300, 8)
    (100, 8)
    (300,)
    (100,)
```

## 4. Обучение моделей

### Функция для подсчета метрик построенных моделей:

```
[43] 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("accuracy: ", accuracy_score(y_test, model.predict(X_test).round()))
      print("balanced_accuracy: ", balanced_accuracy_score(y_test, model.predict(X_test).round()))
```

### Градиентный бустинг с гиперпараметром n = 100:

```
[44] gr_100=GradientBoostingRegressor(n_estimators=100)
      gr_100.fit(X_train, y_train)
```

```
↳ GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                              init=None, learning_rate=0.1, loss='ls', max_depth=3,
                              max_features=None, max_leaf_nodes=None,
                              min_impurity_decrease=0.0, min_impurity_split=None,
                              min_samples_leaf=1, min_samples_split=2,
                              min_weight_fraction_leaf=0.0, n_estimators=100,
                              n_iter_no_change=None, presort='deprecated',
                              random_state=None, subsample=1.0, tol=0.0001,
                              validation_fraction=0.1, verbose=0, warm_start=False)
```

### Метрики данной модели:

```
[45] test_model(gr_100)
```

```
↳ mean_absolute_error: 0.31569576350996903
    median_absolute_error: 0.21688105365467764
    accuracy: 0.75
    balanced_accuracy: 0.7521075873143316
```

### Случайный лес с гиперпараметром n = 100:

```
[46] ran_100 = RandomForestRegressor(n_estimators=100)
      ran_100.fit(X_train, y_train)
```

```
↳ RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                          max_depth=None, max_features='auto', max_leaf_nodes=None,
                          max_samples=None, min_impurity_decrease=0.0,
                          min_impurity_split=None, min_samples_leaf=1,
                          min_samples_split=2, min_weight_fraction_leaf=0.0,
                          n_estimators=100, n_jobs=None, oob_score=False,
                          random_state=None, verbose=0, warm_start=False)
```

## Метрики для оценки:

```
[47] test_model(ran_100)
```

```
↳ mean_absolute_error: 0.297  
   median_absolute_error: 0.205  
   accuracy: 0.72  
   balanced_accuracy: 0.7213970293054998
```

**Вывод:** случайный лес, как оказывается, показывает результат по оценке метрик лучше, чем градиентный бустинг.