



**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 ИУ5 «Системы обработки информации и управления»

Отчёт по РК2

«Технологии машинного обучения»

Вариант 7

Выполнил:

студент группы ИУ5-63Б

Некрасов С. А.

Преподаватель:

Гапанюк Ю. Е.

2023 г.

Задание: Для заданного набора данных (по Вашему варианту) постройте модели классификации или регрессии (в зависимости от конкретной задачи, рассматриваемой в наборе данных). Для построения моделей используйте методы 1 и 2 (по варианту для Вашей группы). Оцените качество моделей на основе подходящих метрик качества (не менее двух метрик). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей? Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д.

Метод №1: Дерево решений

Метод №2: Случайный лес

Датасет: <https://www.kaggle.com/mohansacharya/graduate-admissions> (файл Admission_Predict_Ver1.1.csv)

Решение:

Импортируем необходимые модули и загружаем данные из файла 'Admission_Predict_Ver1.1.csv' в объект `df`. Выводим первые строки данных с помощью `head()` и получаем информацию о данных с помощью `info()`. Удаляем столбец 'Serial No.' и подсчитываем количество пропущенных значений с помощью `isnull().sum()`.

```
In [1]: import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.metrics import mean_absolute_error, mean_squared_error, median_absolute_error, r2_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.impute import SimpleImputer, MissingIndicator
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, MinMaxScaler
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.svm import SVC, NuSVC, LinearSVC
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV
from sklearn import tree
from IPython.core.display import HTML
from sklearn.tree import export_text
from operator import itemgetter
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_predict
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

```
In [2]: df = pd.read_csv('Admission_Predict_Ver1.1.csv')
```

```
In [3]: df.head()
```

```
Out[3]:
```

	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

```
In [4]: df.info()
```

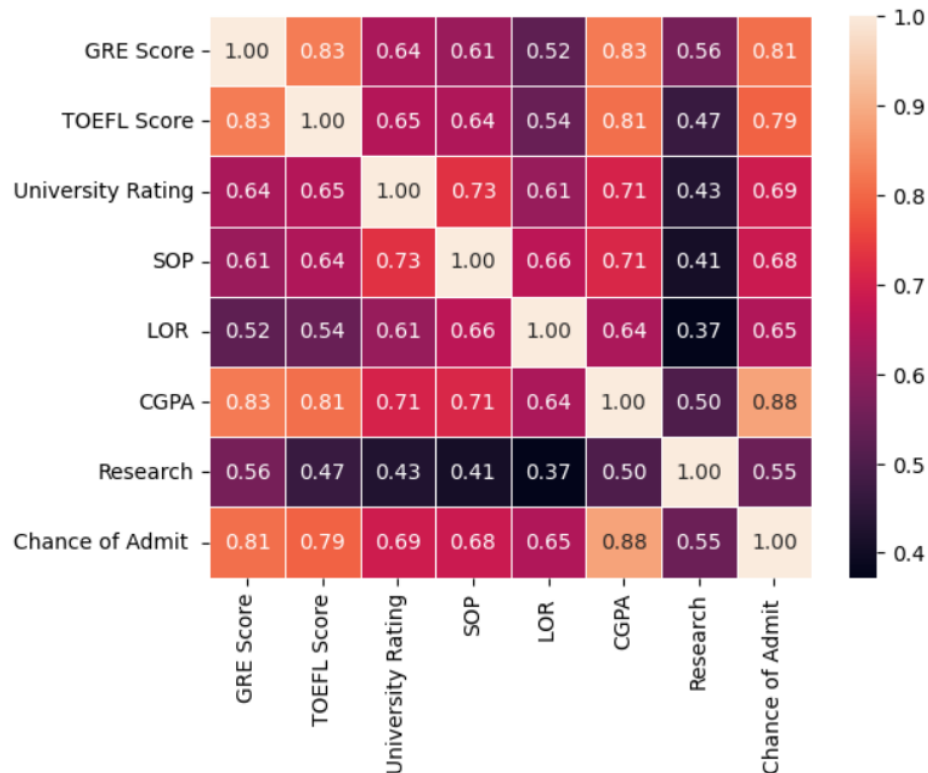
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
#   Column              Non-Null Count  Dtype
---  -
0   Serial No.          500 non-null   int64
1   GRE Score           500 non-null   int64
2   TOEFL Score         500 non-null   int64
3   University Rating   500 non-null   int64
4   SOP                 500 non-null   float64
5   LOR                 500 non-null   float64
6   CGPA                500 non-null   float64
7   Research            500 non-null   int64
8   Chance of Admit     500 non-null   float64
dtypes: float64(4), int64(5)
memory usage: 35.3 KB
```

```
In [5]: df = df.drop(['Serial No.'], axis=1)
df.isnull().sum()
```

```
Out[5]: GRE Score          0
TOEFL Score              0
University Rating        0
SOP                      0
LOR                      0
CGPA                     0
Research                 0
Chance of Admit          0
dtype: int64
```

Создаём матрицу корреляции и визуализируем её с помощью тепловой карты, чтобы оценить связи между признаками в данных.

```
In [6]: corr = df.corr()  
sns.heatmap(corr, linewidths=.5, annot=True, fmt=".2f")  
plt.show()
```



Наиболее сильная корреляция с целевым признаком "Chance of Admit" наблюдается у признаков "CGPA", "GRE Score" и "TOEFL Score". При построении модели машинного обучения эти признаки будут наиболее информативными и важными для прогнозирования результата. Следует отметить наличие корреляции между признаками "SOP" и "University Rating". Это может указывать на взаимосвязь между качеством заявки на поступление (SOP) и рейтингом университета, где студент подает заявку. Можно построить модель машинного обучения, используя признаки "CGPA", "GRE Score", "TOEFL Score", "LOR" и "Research". При этом особенно важными являются первые три признака, так как они сильно коррелируют с результатом.

Далее выполняем удаление столбца 'Chance of Admit', кодируем столбец 'Chance of Admit' из строковых значений в числовые и разбиваем данные на обучающую и тестовую выборки в соотношении 80/20.

```
In [7]: x = df.drop(['Chance of Admit '], axis=1) #Наименования признаков  
y = df['Chance of Admit '] # Значения признаков
```

```
In [8]: # кодируем категориальные данные из строк в числа  
le = LabelEncoder()  
y = le.fit_transform(y)
```

```
In [9]: x_train, x_test, y_train, y_test = train_test_split(X,y,test_size = 0.20, shuffle=False)
```

```
In [10]: # Размер обучающей выборки  
x_train.shape, y_train.shape
```

```
Out[10]: ((400, 7), (400,))
```

```
In [11]: # Размер тестовой выборки  
x_test.shape, y_test.shape
```

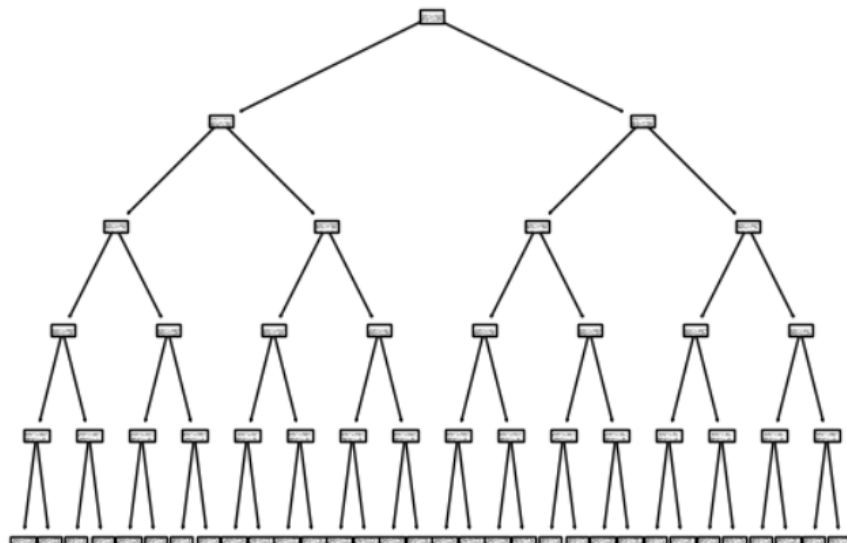
```
Out[11]: ((100, 7), (100,))
```

Затем создаем регрессор `DecisionTreeRegressor` с максимальной глубиной 5 и обучаем его на обучающих данных. Затем мы строим **дерево решений** для визуализации полученной модели. Далее, мы создаем классификатор `DecisionTreeClassifier` с указанным случайным состоянием и обучаем его на обучающих данных. Также определена функция `test_model()` для оценки модели с использованием различных метрик. В конце вызываем функцию `test_model()` для оценки модели `dt_none` на тестовых данных.

```
In [12]: dt_none = DecisionTreeRegressor(max_depth=5)  
dt_none.fit(X_train, y_train)
```

```
Out[12]: DecisionTreeRegressor(max_depth=5)
```

```
In [13]: tree.plot_tree(dt_none);
```



```
In [14]: clf = DecisionTreeClassifier(random_state=1)
         clf.fit(X_train, y_train)
```

```
Out[14]: DecisionTreeClassifier(random_state=1)
```

```
In [15]: 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)))
```

```
In [16]: test_model(dt_none)

mean_absolute_error: 4.4544738590581145
median_absolute_error: 3.3111111111111111
r2_score: 0.7901077601317965
```

Определим функцию `draw_feature_importances`, которая выводит график важности признаков на основе модели дерева решений (`clf`) и набора данных признаков (`X_train`). График отображает столбцы с названиями признаков и их важностью, а также выводит значения важности над соответствующими столбцами.

```
In [17]: def draw_feature_importances(tree_model, X_dataset, figsize=(18, 5)):
         """
         Вывод важности признаков в виде графика
         """
         # Сортировка значений важности признаков по убыванию
         list_to_sort = list(zip(X_dataset.columns.values, tree_model.feature_importances_))
         sorted_list = sorted(list_to_sort, key=itemgetter(1), reverse=True)

         # Названия признаков
         labels = [x for x, _ in sorted_list]
         # Важности признаков
         data = [x for _, x in sorted_list]

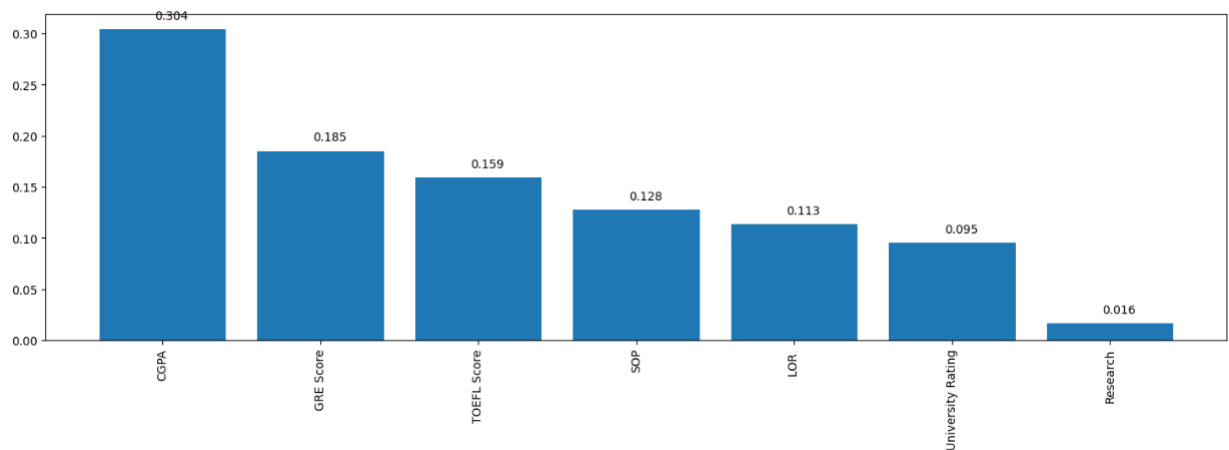
         # Вывод графика
         fig, ax = plt.subplots(figsize=figsize)
         ind = np.arange(len(labels))
         plt.bar(ind, data)
         plt.xticks(ind, labels, rotation='vertical')

         # Вывод значений
         for a, b in zip(ind, data):
             plt.text(a - 0.05, b + 0.01, str(round(b, 3)))

         plt.show()

         return labels, data
```

```
In [18]: dt_fl, dt_fd = draw_feature_importances(clf, X_train)
```



Далее выполняем поиск наилучших параметров для модели `DecisionTreeClassifier` с помощью кросс-валидации и оцениваем ее точность на тестовых данных, а также сравниваем ее с другими моделями.

```
In [19]: tree = DecisionTreeClassifier()

param_grid = {'max_depth': [2, 4, 6, 8, 10],
              'min_samples_split': [2, 4, 6, 8, 10],
              'min_samples_leaf': [1, 2, 3, 4, 5]}
grid_search = GridSearchCV(tree, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
accuracy_tree = grid_search.best_estimator_.score(X_test, y_test)

print("Наилучшие параметры: {} ".format(grid_search.best_params_))
print("Оценка точности на кросс-валидации: {:.2f}".format(grid_search.best_score_))
print(accuracy_tree)
```

```
Наилучшие параметры: {} {'max_depth': 4, 'min_samples_leaf': 3, 'min_samples_split': 10}
Оценка точности на кросс-валидации: 0.10
0.14
```

```
In [20]: models = [['DecisionTree : ', DecisionTreeRegressor()]]
```

```
In [21]: print('Вывод 1')
for name, model in models:
    model = model
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    print(name, (np.sqrt(mean_squared_error(y_test, predictions))))
```

```
Вывод 1
DecisionTree : 9.311820445004296
```

```
In [22]: models = [['DecisionTree:', DecisionTreeRegressor(max_depth=6)],
                  ['линейная регрессия:', LinearRegression(normalize=True)],
                  ['SVC:', SVC(C=1, kernel='linear')]]
```

```
In [23]: print('Вывод 2')
for name,model in models:
    model = model
    model.fit(X_train, y_train)
    predictions = model.predict(X_test)
    print(name, (np.sqrt(mean_squared_error(y_test, predictions))))
```

```
Вывод 2
DecisionTree: 7.257068113982736
линейная регрессия: 4.233037381515526
SVC: 6.495382975621991
```

После создаем модель **случайного леса** ('RandomForestClassifier'), обучаем ее на тренировочных данных, оцениваем ее точность на тестовых данных, а затем выполняем поиск наилучших параметров для модели с помощью кросс-валидации и оцениваем ее точность на тестовых данных с использованием найденных параметров.

```
In [24]: # Создаем модель случайного леса с 100 деревьями
rf_model = RandomForestClassifier(n_estimators=100)
# Обучаем модель на тренировочных данных
rf_model.fit(X_train, y_train)
# Оцениваем качество модели на тестовых данных
accuracy = rf_model.score(X_test, y_test)
print('Accuracy: {:.2f}%'.format(accuracy*100))
```

```
Accuracy: 8.00%
```

```
In [25]: model = RandomForestClassifier()
param_grid = {
    'n_estimators': [200, 700],
    'max_features': ['auto', 'sqrt', 'log2']
}

grid_search = GridSearchCV(model, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)
accuracy_RandomForestClassifier = grid_search.best_estimator_.score(X_test,y_test)

print("Наилучшие параметры: {}", grid_search.best_params_)
print("Оценка точности на кросс-валидации: {:.2f}".format(grid_search.best_score_))
print(accuracy_tree)
```

```
Наилучшие параметры: {} {'max_features': 'sqrt', 'n_estimators': 700}
Оценка точности на кросс-валидации: 0.14
0.14
```