

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



**Отчет**  
**Лабораторная работа № 5**  
**По курсу «Технологии машинного обучения»**

**ИСПОЛНИТЕЛЬ:**

Группа ИУ5-65Б

Камалов М.Р.

"1" июня 2021 г.

**ПРЕПОДАВАТЕЛЬ:**

Гапанюк Ю.Е.

\_\_\_\_\_

"\_\_" \_\_\_\_\_ 2021 г.

Москва 2021

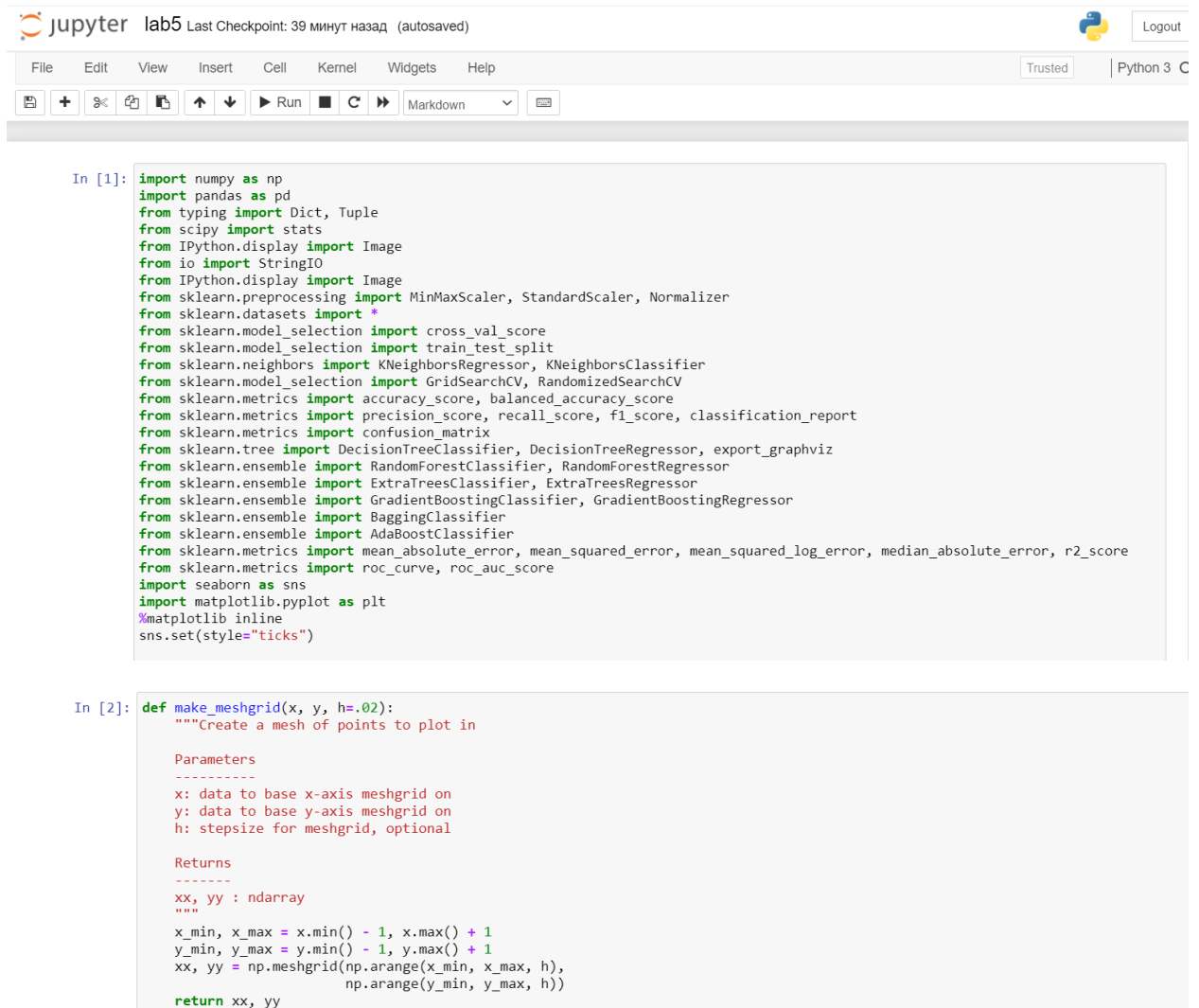
---

**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.

## Скриншоты jupyter notebook



```
In [1]: import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from io import StringIO
from IPython.display import Image
from sklearn.preprocessing import MinMaxScaler, StandardScaler, Normalizer
from sklearn.datasets import *
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")

In [2]: def make_meshgrid(x, y, h=.02):
        """Create a mesh of points to plot in

        Parameters
        -----
        x: data to base x-axis meshgrid on
        y: data to base y-axis meshgrid on
        h: stepsize for meshgrid, optional

        Returns
        -----
        xx, yy : ndarray
        """
        x_min, x_max = x.min() - 1, x.max() + 1
        y_min, y_max = y.min() - 1, y.max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                              np.arange(y_min, y_max, h))
        return xx, yy
```

```
def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

    Parameters
    -----
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    #Можно проверить все ли метки классов предсказываются
    #print(np.unique(Z))
    out = ax.contourf(xx, yy, Z, **params)
    return out


def plot_cl(clf):
    title = clf.__repr__
    clf.fit(iris_X, iris_y)
    fig, ax = plt.subplots(figsize=(5,5))
    X0, X1 = iris_X[:, 0], iris_X[:, 1]
    xx, yy = make_meshgrid(X0, X1)
    plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=iris_y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('Sepal length')
    ax.set_ylabel('Sepal width')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
    plt.show()
```

In [3]:

```
from operator import itemgetter

def draw_feature_importances(tree_model, X_dataset, figsize=(10,5)):
    """
    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 [4]:

```
# Визуализация дерева
def get_png_tree(tree_model_param, feature_names_param):
    dot_data = StringIO()
    export_graphviz(tree_model_param, out_file=dot_data, feature_names=feature_names_param,
                    filled=True, rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph.create_png()
```

```
In [5]: def accuracy_score_for_classes(
        y_true: np.ndarray,
        y_pred: np.ndarray) -> Dict[int, float]:
        """
        Вычисление метрики ассигасу для каждого класса
        y_true - истинные значения классов
        y_pred - предсказанные значения классов
        Возвращает словарь: ключ - метка класса,
        значение - Ассигасу для данного класса
        """
        # Для удобства фильтрации сформируем Pandas DataFrame
        d = {'t': y_true, 'p': y_pred}
        df = pd.DataFrame(data=d)
        # Метки классов
        classes = np.unique(y_true)
        # Результирующий словарь
        res = dict()
        # Перебор меток классов
        for c in classes:
            # отфильтруем данные, которые соответствуют
            # текущей метке класса в истинных значениях
            temp_data_flt = df[df['t']==c]
            # расчет ассигасу для заданной метки класса
            temp_acc = accuracy_score(
                temp_data_flt['t'].values,
                temp_data_flt['p'].values)
            # сохранение результата в словарь
            res[c] = temp_acc
        return res
```

```
In [12]: def accuracy_score_for_classes(
        y_true: np.ndarray,
        y_pred: np.ndarray) -> Dict[int, float]:
        """
        Вычисление метрики ассигасу для каждого класса
        y_true - истинные значения классов
        y_pred - предсказанные значения классов
        Возвращает словарь: ключ - метка класса,
        значение - Ассигасу для данного класса
        """
        # Для удобства фильтрации сформируем Pandas DataFrame
        d = {'t': y_true, 'p': y_pred}
        df = pd.DataFrame(data=d)
        # Метки классов
        classes = np.unique(y_true)
        # Результирующий словарь
        res = dict()
        # Перебор меток классов
        for c in classes:
            # отфильтруем данные, которые соответствуют
            # текущей метке класса в истинных значениях
            temp_data_flt = df[df['t']==c]
            # расчет ассигасу для заданной метки класса
            temp_acc = accuracy_score(
                temp_data_flt['t'].values,
                temp_data_flt['p'].values)
            # сохранение результата в словарь
            res[c] = temp_acc
        return res
```

```
def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики ассигасу для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
        for i in accs:
            print('{} \t {}'.format(i, accs[i]))
```

## Выборка датасета и ее разделение на тестовую и обучающую

```
In [6]: wine = load_wine()
```

```
In [7]: # Сформируем DataFrame
        wine_df = pd.DataFrame(data= np.c_[wine['data']],
                                columns= wine['feature_names'])
```

```
In [8]: wine_df
```

Out[8]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od310
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	
...	...	...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64	
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	7.30	0.70	
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.20	0.59	
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.30	0.60	
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.20	0.61	

178 rows × 13 columns

```
In [9]: sc = MinMaxScaler()
wine_sc = sc.fit_transform(wine_df)
wine_sc
```

```
Out[9]: array([[0.84210526, 0.1916996, 0.57219251, ..., 0.45528455, 0.97069597,
0.56134094],
[0.57105263, 0.2055336, 0.4171123, ..., 0.46341463, 0.78021978,
0.55064194],
[0.56052632, 0.3201581, 0.70053476, ..., 0.44715447, 0.6959707,
0.64693295],
...,
[0.58947368, 0.69960474, 0.48128342, ..., 0.08943089, 0.10622711,
0.39728959],
[0.56315789, 0.36561265, 0.54010695, ..., 0.09756098, 0.12820513,
0.40085592],
[0.81578947, 0.66403162, 0.73796791, ..., 0.10569106, 0.12087912,
0.20114123]])
```

```
In [10]: X_train, X_test, Y_train, Y_test = train_test_split(
wine_sc, wine.target, test_size=0.33, random_state=1)
```

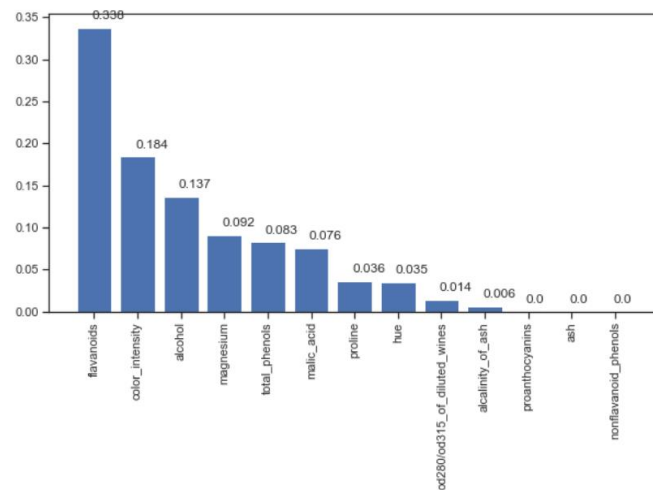
## Обучение моделей и тестирование

### Случайный лес

```
In [23]: # Обучим классификатор на 5 деревьях
tree_wine = RandomForestClassifier(n_estimators=5, oob_score=True, random_state=10)
```

```
In [25]: # Важность признаков
tree_wine.fit(X_train, Y_train)
_ = draw_feature_importances(tree_wine, wine_df)

C:\Users\User\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:540: UserWarning: Some inputs do not have OOB scores. This probably means too few trees were used to compute any reliable oob estimates.
  warn("Some inputs do not have OOB scores. ")
C:\Users\User\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:544: RuntimeWarning: invalid value encountered in true_divide
  decision = (predictions[k] /
```



```
In [15]: target1 = tree_wine.predict(X_test)
accuracy_score(Y_test, target1), precision_score(Y_test, target1, average='macro')
```

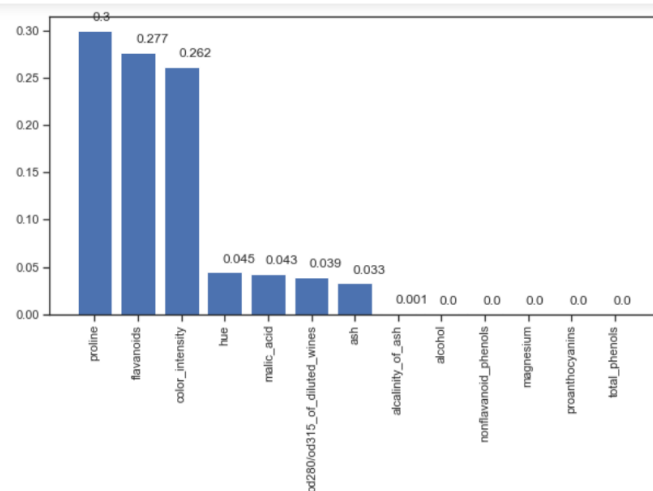
```
Out[15]: (0.9830508474576272, 0.9866666666666667)
```

```
In [16]: print_accuracy_score_for_classes(Y_test, target1)
```

Метка	Accuracy
0	1.0
1	0.9545454545454546
2	1.0

## Бустинг

```
In [13]: # Важность признаков
gr_boost_wine = GradientBoostingClassifier(random_state=1)
gr_boost_wine.fit(X_train, Y_train)
_, _ = draw_feature_importances(gr_boost_wine, wine_df)
```



```
In [17]: target2 = gr_boost_wine.predict(X_test)
accuracy_score(Y_test, target2), precision_score(Y_test, target2, average='macro')
```

```
Out[17]: (0.9661016949152542, 0.9743589743589745)
```

```
In [19]: print_accuracy_score_for_classes(Y_test, target2)
```

Метка	Accuracy
0	1.0
1	0.9090909090909091
2	1.0

## Выводы

Принимая во внимание, что модель **случайного леса** получила результат точнее, можем сделать вывод, что датасет wine содержит в основном простые зависимости, нежели сложные. Это означает, что борьба с переобучением приносит лучшие результаты в этом датасете.