Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

Дисциплина «Технологии машинного обучения»

Отчёт

по лабораторной работе №6

«Ансамбли моделей машинного обучения»

Вариант 12

Студент:

Крюков Г. М.

Группа ИУ5-61Б

Преподаватель:

Гапанюк Ю. Е.

Москва, 2020 г.

Цель лабораторной работы:

Изучение ансамблей моделей машинного обучения.

Задание:

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
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 работы:

Выбранный датасет:

https://www.kaggle.com/ronitf/heart-disease-uci/data#

Датасет содержит 76 атрибутов, но все опубликованные эксперименты относятся к использованию подмножества из 14 из них. Поле "цель" относится к наличию у пациента сердечно-сосудистых заболеваний. Это целочисленное значение от 0 (отсутствие присутствия) до 4.

Attribute Information:

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholestoral in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)
8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

Подготовка данных:

```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from sklearn.externals.six import StringIO
from IPython.display import Image
import graphviz
import pydotplus
from sklearn.datasets import load_iris, load_boston
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")
```

```
[8]  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
```

```
[9]  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
```

```
# 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()
```

```
[11] def accuracy_score_for_classes(
        y_true: np.ndarray,
        y_pred: np.ndarray) -> Dict[int, float]:
        """
        Вычисление метрики accuracy для каждого класса
        y_true - истинные значения классов
        y_pred - предсказанные значения классов
        Возвращает словарь: ключ - метка класса,
        значение - Accuracy 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 accuracy 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 accuracy 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]))
```

Загрузка данных

```
data=pd.read_csv("/heart.csv")
data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Разделение данных на обучающую и тестовую выборки

```
[3]  target = data['target']
     data = data.drop('target', axis = 1)
```

```
[4]  #Разделим данные на тестовую и обучающую выборку
     X_train, X_test, y_train, y_test = train_test_split(
         data, target, test_size=0.2, random_state=1)
```

1. **Бэггинг**

```
[12] # Обучим классификатор на 5 деревьях
     bc1 = BaggingClassifier(n_estimators=5, oob_score=True, random_state=10)
     bc1.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/_bagging.py:633: UserWarning: Some inpu
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/_bagging.py:638: RuntimeWarning: invali
  predictions.sum(axis=1)[:, np.newaxis])
BaggingClassifier(base_estimator=None, bootstrap=True, bootstrap_features=False,
                  max_features=1.0, max_samples=1.0, n_estimators=5,
                  n_jobs=None, oob_score=True, random_state=10, verbose=0,
                  warm_start=False)
```

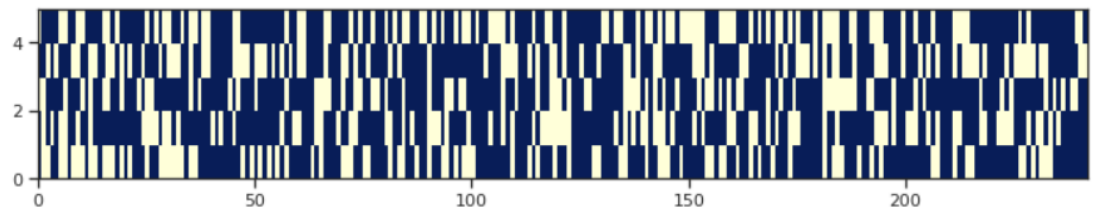
[13]
```
# Какие объекты были использованы в обучающей выборке каждого дерева
bc1.estimators_samples_
```

```
[array([165, 137, 177, 180, 103, 142, 138,  26, 202, 152, 238, 138, 191,
        179,  50, 172, 126, 240,  67,  34, 221,  24,  43, 187, 149, 230,
        228, 236, 186,  58, 112, 118, 104,  46, 104, 203,  27,  74, 147,
         37, 228,  45, 187, 182, 132,  44, 142,  69, 156, 203, 225, 235,
         74, 234,  23, 167, 108, 188,  64, 171,   0,  50, 201, 150,  78,
        171,  42, 112,  77, 156,  50,   4, 114,  14,  56, 170, 196, 105,
        204,  43,  39, 196, 191,  43, 139, 212, 223, 237,  80, 176, 127,
        159, 116, 225,  56,  54, 199, 178, 210, 110, 138, 237, 220, 136,
        205,   4,  79, 236,  62,  44,  60, 218, 111,  74, 153, 114, 125,
        195, 137, 197, 102, 153,  88,  14, 130, 107, 110, 175, 118,  41,
        219, 151, 174,  62, 208,  66,  37,  14,  52, 120, 117, 224, 171,
        209,  68, 176, 171, 216, 213,  73, 190,  39, 188, 104,  92, 198,
        150,  44, 139, 165,  22, 167,  66, 163, 107, 171,  27, 241, 224,
        238, 153,  85,  54,  40, 146, 222, 236, 234,  95,  38, 168,  92,
         97, 186, 202,  61, 180, 219, 116,  73, 187, 199, 116,  68,  48,
         20, 124,  82, 180,  37,  58, 101,   7, 123, 141, 202, 146,  38,
        116, 105,  91, 163, 239, 193, 200,   7, 218,   0, 131,   3,  22,
        167,  59, 236, 133,  20, 106, 162, 123,  11, 121,  66,  18, 182,
         46,  52, 147, 160,  62,  89, 217,  86]),
 array([ 95, 215,  46, 226, 234,  93, 129,  44, 136, 226,  87, 149,  61,
         68,  87, 183, 238, 102,  31,  34,  17,  58, 162,  38,  79, 149,
        208,  88, 166, 237,  70,  22,  88, 112, 115, 237, 167,  52, 152,
        147, 172,  19, 158,  15,  49, 178,  39, 180,  41, 130,  14,  47,
        161, 127,  18,  13, 138,  13, 236, 190, 102, 152,  15, 191,  57,
        111, 145, 176,  62, 180, 227,  23, 180, 128, 215,  16,  67,  50,
        115, 187,  82,  43,  55,   7, 210, 105,   4,  51, 102,  98,  35,
        124, 206,  52,  35, 191, 209,  58,  67, 148, 240, 225, 209, 182,
```

[21]
```
# Сконвертируем эти данные в двоичную матрицу,
# 1 соответствует элементам, попавшим в обучающую выборку
bin_array = np.zeros((5, X_train.shape[0]))
for i in range(5):
    for j in bc1.estimators_samples_[i]:
        bin_array[i][j] = 1
    print(bin_array)
```

```
[[1. 0. 0. ... 1. 1. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[1. 0. 0. ... 1. 1. 1.]
 [1. 0. 1. ... 1. 1. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[1. 0. 0. ... 1. 1. 1.]
 [1. 0. 1. ... 1. 1. 1.]
 [0. 0. 1. ... 0. 1. 1.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[1. 0. 0. ... 1. 1. 1.]
 [1. 0. 1. ... 1. 1. 1.]
 [0. 0. 1. ... 0. 1. 1.]
 [1. 1. 0. ... 1. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
[[1. 0. 0. ... 1. 1. 1.]
 [1. 0. 1. ... 1. 1. 1.]
 [0. 0. 1. ... 0. 1. 1.]
 [1. 1. 0. ... 1. 0. 0.]
```

[22]
```
# И визуализируем (синим цветом показаны данные, которые попали в обучающую выборку)
fig, ax = plt.subplots(figsize=(12,2))
ax.pcolor(bin_array, cmap='YlGnBu')
plt.show()
```



[23]
```
# Оценим Out-of-bag error, теоретическое значение 37%
for i in range(5):
    cur_data = bin_array[i]
    len_cur_data = len(cur_data)
    sum_cur_data = sum(cur_data)
    (len(bin_array[0]) - sum(bin_array[0])) / len(bin_array[0])
    oob_i = (len_cur_data - sum_cur_data) / len_cur_data
    print('Для модели № {} размер OOB составляет {}%'.format(i+1, round(oob_i, 4)*100.0))
```

```
Для модели № 1 размер OOB составляет 36.36%
Для модели № 2 размер OOB составляет 36.36%
Для модели № 3 размер OOB составляет 33.47%
Для модели № 4 размер OOB составляет 38.84%
Для модели № 5 размер OOB составляет 39.67%
```

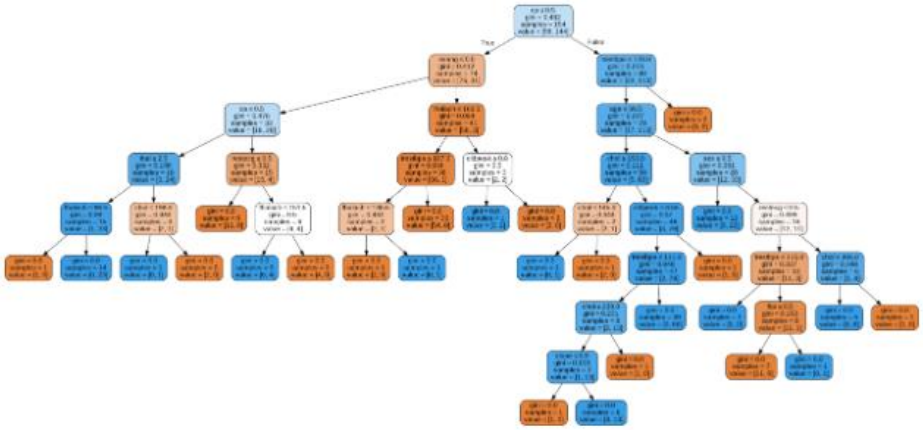
[24]
```
# Out-of-bag error, возвращаемый классификатором
bc1.oob_score_, 1-bc1.oob_score_
```

```
(0.7148760330578512, 0.2851239669421488)
```

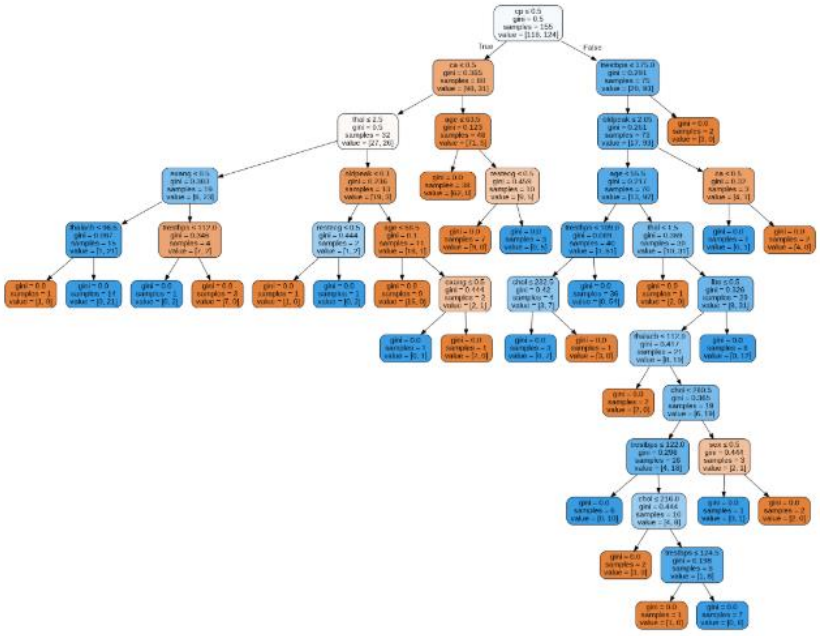
[25]
```
# Параметр oob_decision_function_ возвращает вероятности
# принадлежности объекта к классам на основе oob
# В данном примере три класса,
# значения nan могут возвращаться в случае маленькой выборки
bc1.oob_decision_function_[55:70]
```

```
array([[1.        , 0.        ],
       [0.5       , 0.5       ],
       [0.66666667, 0.33333333],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.33333333, 0.66666667],
       [1.        , 0.        ],
       [       nan,        nan],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [0.5       , 0.5       ],
       [       nan,        nan],
       [0.        , 1.        ]])
```

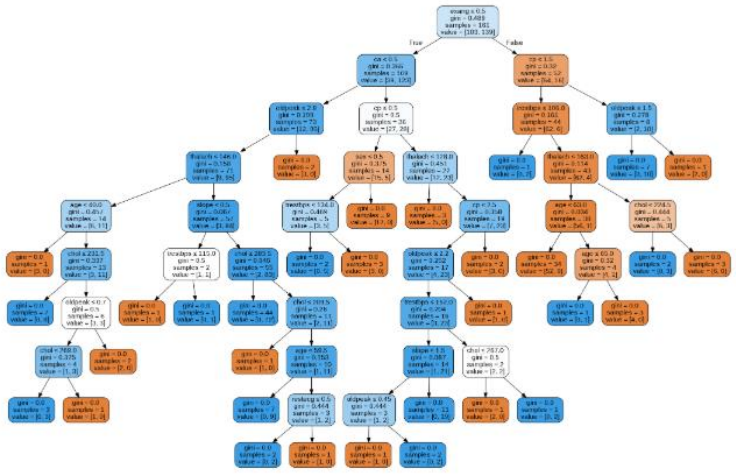
```
[35] # Визуализация обученных решающих деревьев
     Image(get_png_tree(bc1.estimators_[0], X_train.columns), width="500")
```



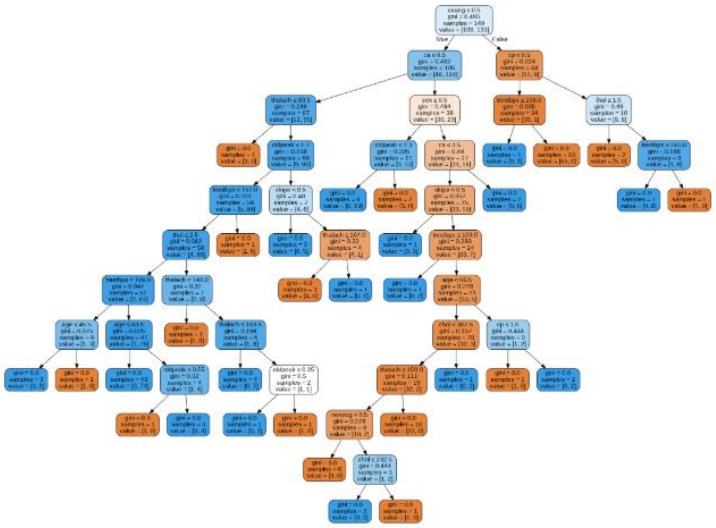
```
[36] Image(get_png_tree(bc1.estimators_[1], X_train.columns), width="500")
```



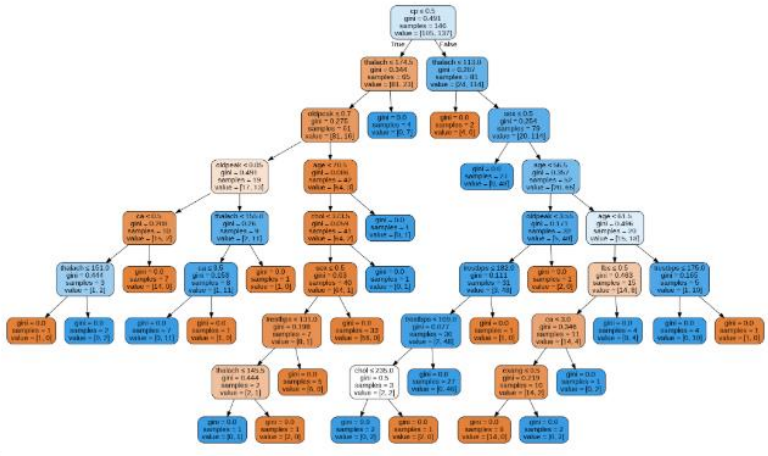
```
[37] Image(get_png_tree(bc1.estimators_[2], X_train.columns), width="500")
```



```
[38] Image(get_png_tree(bc1.estimators_[3], X_train.columns), width="500")
```



```
Image(get_png_tree(bc1.estimators_[4], X_train.columns), width="500")
```



Деревья получаются различными. Таким образом, каждое дерево работает как "слабая модель".

2. Случайный лес

```
[44] # Обучим классификатор на 5 деревьях
     tree1 = RandomForestClassifier(n_estimators=5, oob_score=True, random_state=10)
     tree1.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/_forest.py:523: UserWarning
  warn("Some inputs do not have OOB scores. "
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/_forest.py:528: RuntimeWarn
  predictions[k].sum(axis=1)[:, np.newaxis])
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', 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=5,
                       n_jobs=None, oob_score=True, random_state=10, verbose=0,
                       warm_start=False)
```

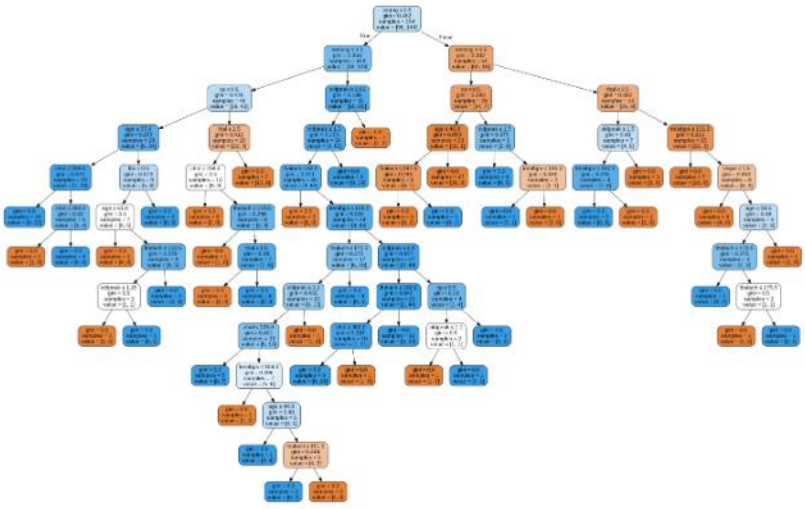
```
[45] # Out-of-bag error, возвращаемый классификатором
     tree1.oob_score_, 1-tree1.oob_score_
```

```
(0.7024793388429752, 0.2975206611570248)
```

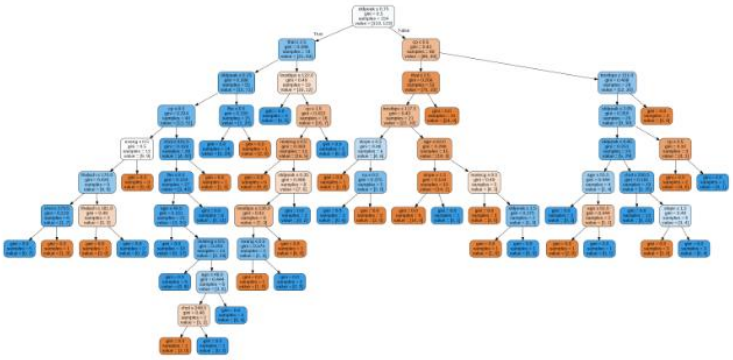
```
[46] tree1.oob_decision_function_[55:70]
```

```
array([[0.        , 1.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [0.33333333, 0.66666667],
       [1.        , 0.        ],
       [       nan,        nan],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [1.        , 0.        ],
       [0.        , 1.        ],
       [1.        , 0.        ],
       [       nan,        nan],
       [0.        , 1.        ]])
```

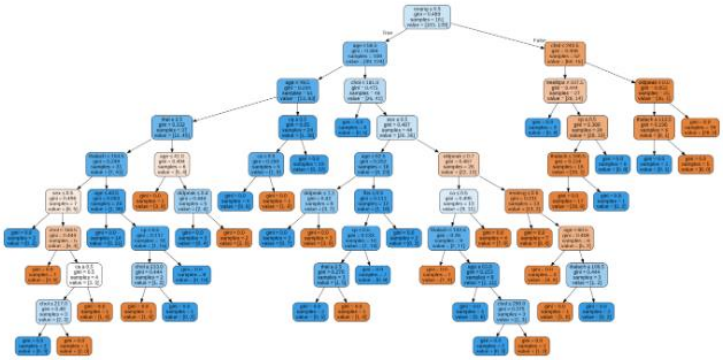
```
Image(get_png_tree(tree1.estimators_[0], X_train.columns), width="500")
```



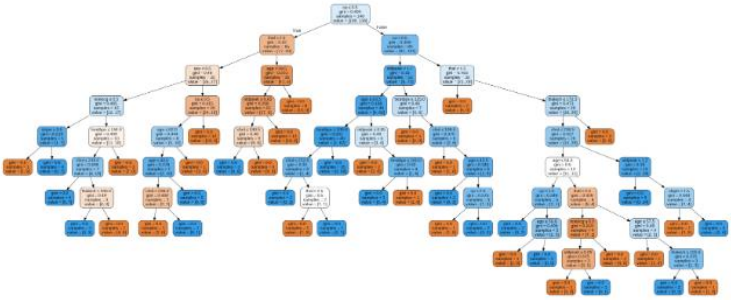
```
[48] Image(get_png_tree(tree1.estimators_[1], X_train.columns), width="500")
```



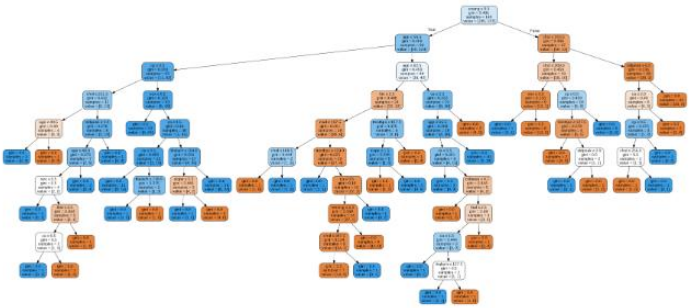
```
[49] Image(get_png_tree(tree1.estimators_[2], X_train.columns), width="500")
```



```
[50] Image(get_png_tree(tree1.estimators_[3], X_train.columns), width="500")
```



```
Image(get_png_tree(tree1.estimators_[4], X_train.columns), width="500")
```



В случае случайного леса деревья получаются более разнообразными, чем в случае бэггинга.

Сравнение моделей

Метрики качества классификации

1) Accuracy

Метрика вычисляет процент (долю в диапазоне от 0 до 1) правильно определенных классов.

2) Метрика precision:

Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

```
[52] target_bagging = bc1.predict(X_test)

     accuracy_score(y_test, target_bagging), \
     precision_score(y_test, target_bagging)
```

```
(0.7377049180327869, 0.7419354838709677)
```

```
[53] target_tree= tree1.predict(X_test)

     accuracy_score(y_test, target_tree), \
     precision_score(y_test, target_tree)
```

```
(0.7704918032786885, 0.7428571428571429)
```

Вывод:

Можем видеть, что обе модели по рассматриваемым метрикам показали приемлемый результат, но модель случайного леса оказалась результативнее, получив меньший процент ошибок.