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»

ОТЧЁТ ПО
Лабораторной работе №5

Выполнил:	Проверил:
студент группы ИУ5-65Б	преподаватель каф. ИУ5
Кулешова И. А.	Гапанюк Ю. Е.
Подпись и дата:	Подпись и дата:

Москва

2024

```
In [1]: import numpy as np
import pandas as pd
import sklearn
import seaborn as sns
import matplotlib.pyplot as plt
from typing import Dict, Tuple
from scipy import stats
from sklearn import datasets
from sklearn import model_selection
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_square
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import SGDClassifier
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSV
from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from io import StringIO
from IPython.display import Image
import graphviz
import pydotplus
from sklearn.model_selection import cross_val_score
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, classificat
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_g
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegress
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_square
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(wine_X, wine_y)
    fig, ax = plt.subplots(figsize=(5,5))
    X0, X1 = wine_X[:, 0], wine_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=wine_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_importa
    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')
```

```python
    # 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]:
```python
# 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_n
                    filled=True, rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph.create_png()
```

In [5]:
```python
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 и первичный анализ данных

In [6]:
```
wine = load_wine()
wine_X = wine.data[:, :2]
wine_y = wine.target
```

Разделение выборки на обучающую и тестовую

In [7]:
```
wine_X_train, wine_X_test, wine_y_train, wine_y_test = train_test_split(
    wine.data, wine.target, test_size=0.3, random_state=1)
```

1. Бэггинг

In [8]:
```
# Обучим классификатор на 5 деревьях
bc1 = BaggingClassifier(n_estimators=5, oob_score=True, random_state=10)
bc1.fit(wine_X, wine_y)
```

```
C:\Users\user\anaconda3\Lib\site-packages\sklearn\ensemble\_bagging.py:789: UserW
arning: Some inputs do not have OOB scores. This probably means too few estimator
s were used to compute any reliable oob estimates.
  warn(
C:\Users\user\anaconda3\Lib\site-packages\sklearn\ensemble\_bagging.py:795: Runti
meWarning: invalid value encountered in divide
  oob_decision_function = predictions / predictions.sum(axis=1)[:, np.newaxis]
```

Out[8]:
```
▾ BaggingClassifier
BaggingClassifier(n_estimators=5, oob_score=True, random_state=10)
```

In [9]:
```
# Какие объекты были использованы в обучающей выборке каждого дерева
bc1.estimators_samples_
```

```
Out[9]: [array([165, 137, 177, 103, 142, 138,  26, 152, 138,  50, 172, 126,  67,
          34,  24,  43, 149,  58, 112, 118, 104,  46, 104,  27,  74, 147,
          37,  45, 132,  44, 142,  69, 156,  74,  23, 167, 108,  64, 171,
           0,  50, 150,  78, 171,  42, 112,  77, 156,  50,   4, 114,  14,
          56, 170, 105,  43,  39,  43, 139,  80, 176, 127, 159, 116,  56,
          54, 110, 138, 136,   4,  79,  62,  44,  60, 111,  74, 153, 114,
         125, 137, 102, 153,  88,  14, 130, 107, 110, 175, 118,  41, 151,
         174,  62,  66,  37,  14,  52, 120, 117, 171,  68, 176, 171,  73,
          39, 104,  92, 150,  44, 139, 165,  22, 167,  66, 163, 107, 171,
          27, 153,  85,  54,  40, 146,  95,  38, 168,  92,  97,  61, 116,
          73, 116,  68,  48,  20, 124,  82,  37,  58, 101,   7, 123, 141,
         146,  38, 116, 105,  91, 163,   7,   0, 131,   3,  22, 167,  59,
         133,  20, 106, 162, 123,  11, 121,  66,  18,  46,  52, 147, 160,
          62,  89,  86,  37, 115, 132,  38,  90,  84]),
 array([ 95,  46,  93, 129,  44, 136,  87, 149,  61,  68,  87, 102,  31,
          34,  17,  58, 162,  38,  79, 149,  88, 166,  70,  22,  88, 112,
         115, 167,  52, 152, 147, 172,  19, 158,  15,  49,  39,  41, 130,
          14,  47, 161, 127,  18,  13, 138,  13, 102, 152,  15,  57, 111,
         145, 176,  62,  23, 128,  16,  67,  50, 115,  82,  43,  55,   7,
         105,   4,  51, 102,  98,  35, 124,  52,  35,  58,  67, 148,  49,
         177, 103, 169, 110,  16, 123,  13,  67, 125,  20,  33, 100,   0,
          63, 150,  20, 171,  79, 157,   4, 115, 100,  18, 156, 156, 131,
          78,   7,  62, 141, 103,   7,  75, 101, 108,  21, 155,  48,  28,
          71,   4,  66, 154,  85,  54,  46, 112,  74, 125, 126, 125, 152,
          97,  17, 154,  10,  52,  53,  21,   2, 113,  95,  47, 154,  89,
         103,  54, 136,  43,   8,  19, 134,  36,  38, 106,  37,  83, 172,
          19, 176,  16, 132, 115, 176, 127, 163, 100,  74,  39,  11, 167,
         152,  76,  77, 127,  96, 115, 166,  21,  28]),
 array([153,  54, 131,  76, 112, 169,  99,  54,  92,  55,  38, 141,  40,
          18,  29,  42,  61,  54, 173,  21,  42,  82,   7,  28, 111,  72,
          71,  51,  81, 124,  24,  39, 123, 129,  53, 173, 165,  27,  58,
         125,  83, 106,  34, 139, 141,  17, 151, 114, 113,  85, 124,   7,
           7, 164,  20,   2, 128, 175, 149,  32, 177,  56, 146, 175,   5,
          61,  30, 165, 116, 105,  79,  54, 117,  49,  40, 173,   8,  30,
         121,  87,  48, 150,  18, 103, 101,  58, 134, 146,  45,  33,  55,
          43,  27,  27, 143,  85,  31,  13, 167,  29, 102, 133, 158, 141,
          63, 146, 109,  99,  91, 148, 165,  11, 145,  47,   3, 128, 108,
          59, 129,   9,  79,  41,  22,  85,  21,  58, 166, 148, 104,  86,
         100,  60,  39, 160,  68,  14,  13,   3, 121, 151,  89, 148,  21,
         132,  22, 128,  69, 145,  56,  33, 117,  52, 131,  58, 150,   4,
          11,  60, 153, 175,  62,  75,  49,  43,  88, 170, 163,  93, 176,
          96,  30,   7,  33,  36, 130,  93, 172,  54]),
 array([148, 104,  86,  55,  15, 166, 168, 120, 156, 148,  97,  87, 167,
          85,  34,  67, 145, 164,  42, 174,  98, 114,  44,  88,   7, 127,
         112,  18, 104, 120, 160,  68,  20,  94,  14,   5, 101, 102, 130,
          29,  53,  75, 111, 132,  22,  91,  26,  42,   5, 140, 150,  65,
         100,  55,  55, 161,  94,  84, 139, 173,  78,  37,  20,  94,  59,
          82, 133, 149,  41, 104,  43,  95,  96,  69, 106,  99, 137, 111,
         159,  84,  64,  41,  68, 147, 169,  27,  85,  63, 149, 105, 123,
         122,  80,  62,  36,   1,  96, 134,  93, 160,  68,   7,   6, 173,
          77,  25, 137,  23, 123, 146,  22,  98,   3, 145, 171, 128,  57,
         111,  91,  79,   0, 167,  37,  72, 171,   3,  93,  44,  15, 145,
         105, 147,  65, 159,  95, 154,  87,  92, 132, 165, 173,   3, 130,
         158,  36, 122, 146,  72, 116, 105,  75, 131,  49,  49, 141, 126,
          89, 130,  10, 169,  40, 173,  93,  33,  74,  15,  11,  72, 144,
         157,  15, 124, 117,  78, 111,  55,  22,  49]),
 array([144,  14, 177, 119,  31, 169,  54,  51,  32, 147, 139,  95,  37,
         112, 103,   8, 169,  24, 113, 167,  68,  62, 139,  13,  11, 128,
         112,  42, 126,  17,  63, 155,  71,  17, 142, 139, 127,  56,  62,
          94,  20,  73,  81,  90,  87,  62,  90, 172, 132,   8,  40,   1,
```

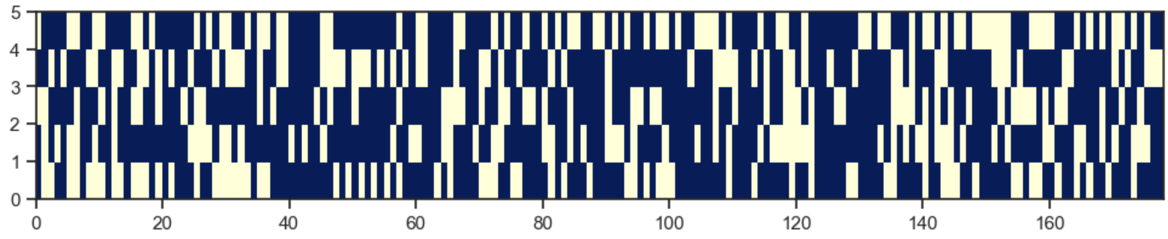
```
          8,  43,   4, 144,  44, 136, 144,  47,  24,  73, 116, 162, 112,
         78, 135,  86, 129,  81,  92, 120, 100,  53, 125,  49, 108,  97,
        156, 126, 155, 111,  21, 123,  22,  21,  83,  23, 122,  95, 124,
        136,  41, 103,  78, 154,  12, 129, 162, 165,   3,  11,  94,  64,
        169,  87,  95, 100,  24,  70, 137,   3, 125,  69,  28,  49,  14,
          7,  22, 155,  32, 115, 177,  37, 116,  70,  32,  34,  83,  97,
         28, 161,   4, 163, 110,  79, 147,  19, 108,  86, 112,  79,  76,
        124, 174,  68, 176, 126,  52,  44,  28,   2, 116,  55, 127,  26,
         56, 111,  71,  65,  87, 155,  32,  79,  19,  79,  59,  48,   1,
        139, 102, 120,  50,  58,  75, 128, 177,  87])]
```

In [10]:

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

```
Out[10]: array([[1., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0.,
                 0., 0., 1., 0., 1., 0., 1., 1., 1., 0., 1., 1., 0., 0., 0., 0.,
                 0., 0., 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,
                 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 1., 1., 1., 1., 0.,
                 1., 0., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1.,
                 1., 0., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 0., 0., 1.,
                 0., 1., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1.,
                 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1.,
                 0., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 0., 1., 1., 0.,
                 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1.,
                 1., 0., 1., 1., 0., 1., 0., 1., 1., 0., 1., 1., 1., 0., 1., 1.,
                 1., 1.],
                [1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1.,
                 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0., 1.,
                 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0., 1., 1.,
                 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1.,
                 0., 0., 1., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1.,
                 0., 0., 1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1.,
                 1., 1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 0., 1., 0., 1., 1.,
                 1., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.,
                 1., 1., 1., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 0., 0.,
                 0., 1., 0., 1., 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 0.,
                 0., 1., 1., 1., 0., 0., 1., 1., 0., 1., 0., 1., 1., 0., 0., 0.,
                 1., 1.],
                [0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1., 0., 1., 1., 0.,
                 0., 1., 1., 0., 1., 1., 1., 0., 1., 0., 0., 1., 1., 1., 1., 1.,
                 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1.,
                 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1.,
                 0., 0., 0., 0., 1., 1., 0., 1., 1., 0., 0., 1., 1., 0., 0., 1.,
                 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 0.,
                 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1.,
                 1., 1., 1., 0., 1., 1., 0., 0., 0., 1., 0., 1., 1., 1., 0., 0.,
                 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 1.,
                 0., 1., 1., 0., 1., 1., 1., 1., 0., 1., 0., 0., 0., 0., 1., 0.,
                 1., 0., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 0., 1.,
                 1., 1.],
                [1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1.,
                 0., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1., 1., 0., 1., 0., 0.,
                 0., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0.,
                 0., 1., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1.,
                 1., 1., 0., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1.,
                 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1.,
                 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0., 1.,
                 1., 0., 1., 0., 1., 1., 0., 0., 1., 0., 1., 1., 1., 0., 1., 1.,
                 1., 0., 1., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 1., 0., 0.,
                 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 1., 0., 1., 1., 1., 1.,
                 1., 1., 0., 0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0.,
                 0., 0.],
                [0., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 0.,
                 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 0., 0., 1.,
                 1., 0., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 1.,
                 1., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0., 1., 1.,
                 1., 1., 0., 0., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0., 1., 1.,
                 0., 1., 0., 1., 0., 0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 1.,
                 0., 1., 0., 0., 1., 0., 1., 1., 0., 0., 0., 0., 1., 0., 1., 1.,
                 1., 1., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1.,
                 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 0., 1., 0., 0., 1., 0.,
                 1., 0., 0., 1., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 0., 0.,
                 0., 1., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 0., 1., 0.,
                 1., 1.]])
```

In [11]:
```
# И визуализируем (синим цветом показаны данные, которые попали в обучающую выбо
fig, ax = plt.subplots(figsize=(12,2))
ax.pcolor(bin_array, cmap='YlGnBu')
plt.show()
```



In [12]:
```
# 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
```

```
Для модели № 1 размер OOB составляет 37.64%
Для модели № 2 размер OOB составляет 35.96%
Для модели № 3 размер OOB составляет 34.27%
Для модели № 4 размер OOB составляет 37.08%
Для модели № 5 размер OOB составляет 42.13%
```

In [13]:
```
# Out-of-bag error, возвращаемый классификатором
# Для классификации используется метрика accuracy
bc1.oob_score_, 1-bc1.oob_score_
```

Out[13]:
```
(0.7528089887640449, 0.2471910112359551)
```

In [14]:
```
# 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]
```

Out[14]:
```
array([[1.        , 0.        , 0.        ],
       [1.        , 0.        , 0.        ],
       [0.66666667, 0.        , 0.33333333],
       [1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 0.5       , 0.5       ],
       [       nan,        nan,        nan],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [       nan,        nan,        nan],
       [0.        , 1.        , 0.        ]])
```

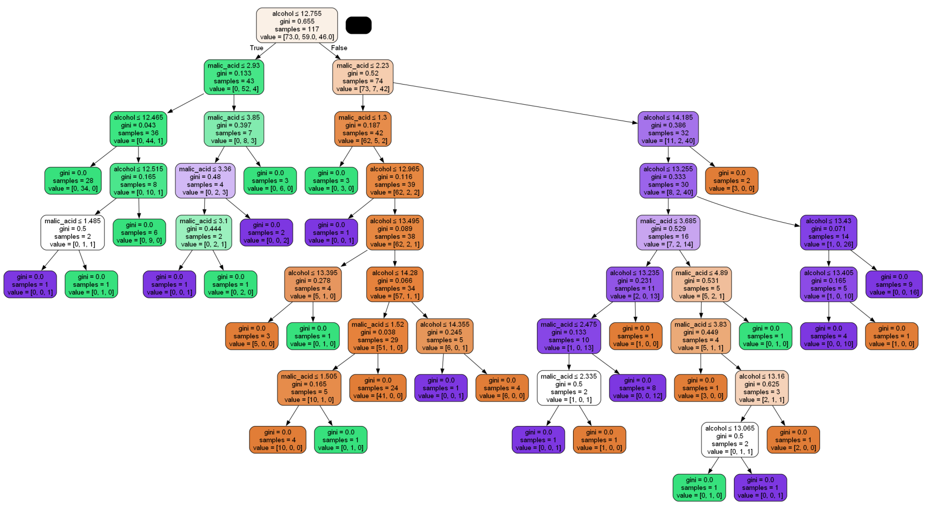
In [15]:
```
# Визуализация обученных решающих деревьев
Image(get_png_tree(bc1.estimators_[0], wine.feature_names[:2]), width='80%')
```

```
---------------------------------------------------------------------------
InvocationException                       Traceback (most recent call last)
Cell In[15], line 2
      1 # Визуализация обученных решающих деревьев
----> 2 Image(get_png_tree(bc1.estimators_[0], wine.feature_names[:2]), width='80%')

Cell In[4], line 7, in get_png_tree(tree_model_param, feature_names_param)
      4 export_graphviz(tree_model_param, out_file=dot_data, feature_names=feature_names_param,
      5                 filled=True, rounded=True, special_characters=True)
      6 graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
----> 7 return graph.create_png()

File ~\anaconda3\Lib\site-packages\pydotplus\graphviz.py:1797, in Dot.__init__.<locals>.<lambda>(f, prog)
   1792 # Automatically creates all the methods enabling the creation
   1793 # of output in any of the supported formats.
   1794 for frmt in self.formats:
   1795     self.__setattr__(
   1796         'create_' + frmt,
-> 1797         lambda f=frmt, prog=self.prog: self.create(format=f, prog=prog)
   1798     )
   1799     f = self.__dict__['create_' + frmt]
   1800     f.__doc__ = (
   1801         '''Refer to the docstring accompanying the'''
   1802         ''''create' method for more information.'''
   1803     )

File ~\anaconda3\Lib\site-packages\pydotplus\graphviz.py:1959, in Dot.create(self, prog, format)
   1957     self.progs = find_graphviz()
   1958     if self.progs is None:
-> 1959         raise InvocationException(
   1960             'GraphViz\'s executables not found')
   1962 if prog not in self.progs:
   1963     raise InvocationException(
   1964         'GraphViz\'s executable "%s" not found' % prog)

InvocationException: GraphViz's executables not found
```

In [ ]:
```
Image(get_png_tree(bc1.estimators_[1], wine.feature_names[:2]), width='80%')
```

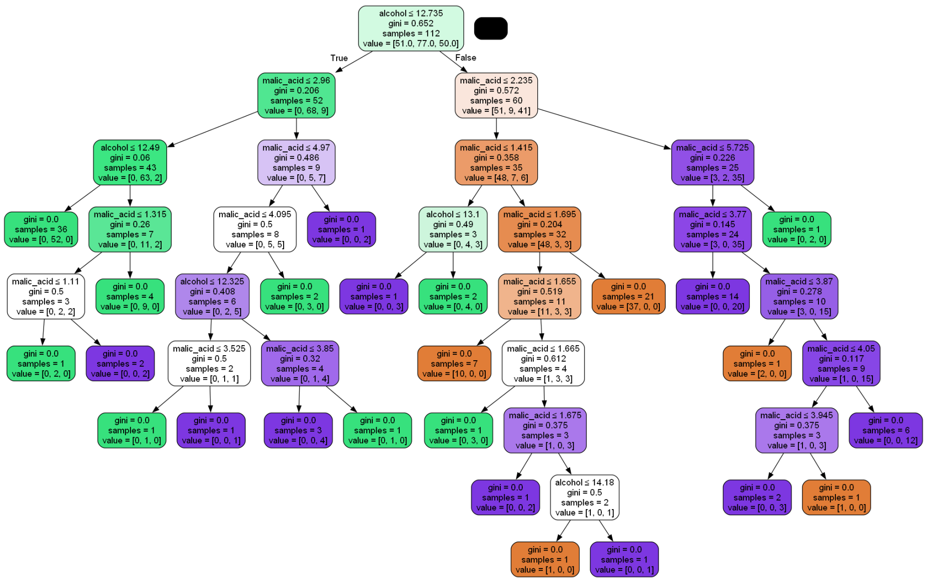
In [75]:
```
Image(get_png_tree(bc1.estimators_[2], wine.feature_names[:2]), width='80%')
```

Out[75]:



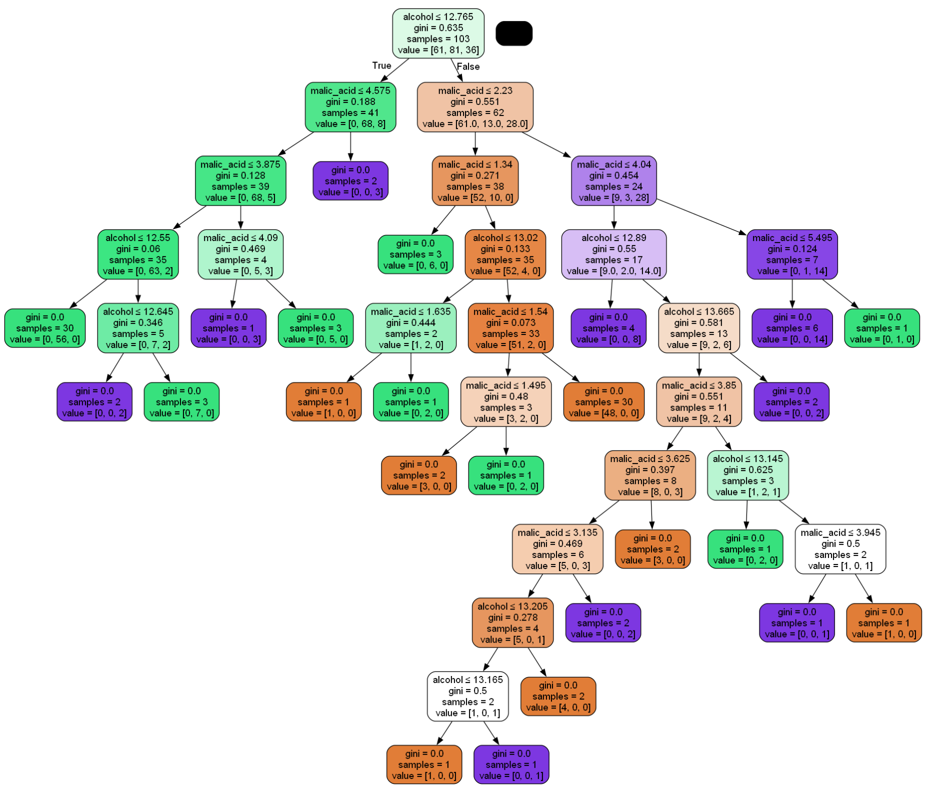
In [76]:
```
Image(get_png_tree(bc1.estimators_[3], wine.feature_names[:2]), width='80%')
```

Out[76]:



In [77]:
```
Image(get_png_tree(bc1.estimators_[4], wine.feature_names[:2]), width='80%')
```

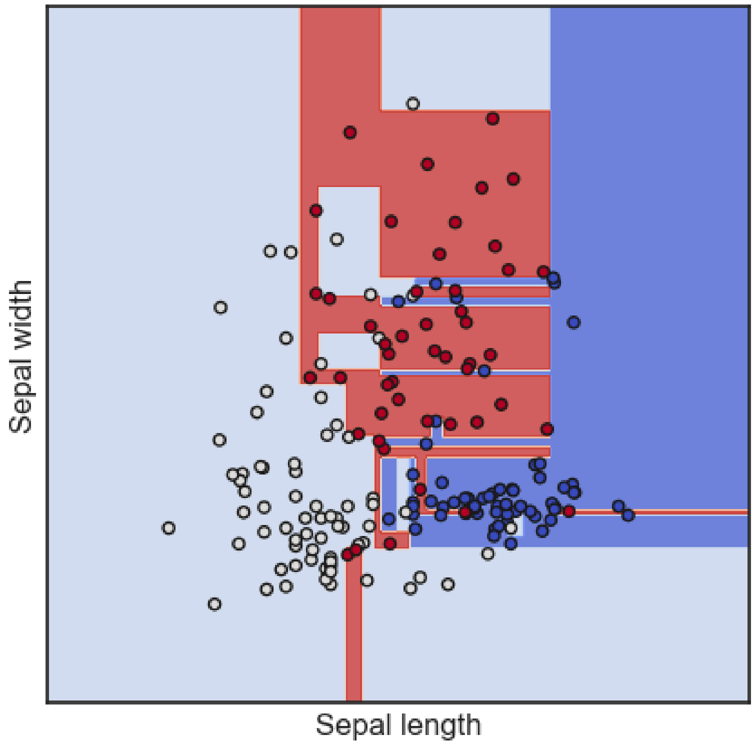
Out[77]:



Визуализация результатов

In [78]:
```
plot_cl(DecisionTreeClassifier(random_state=1))
```

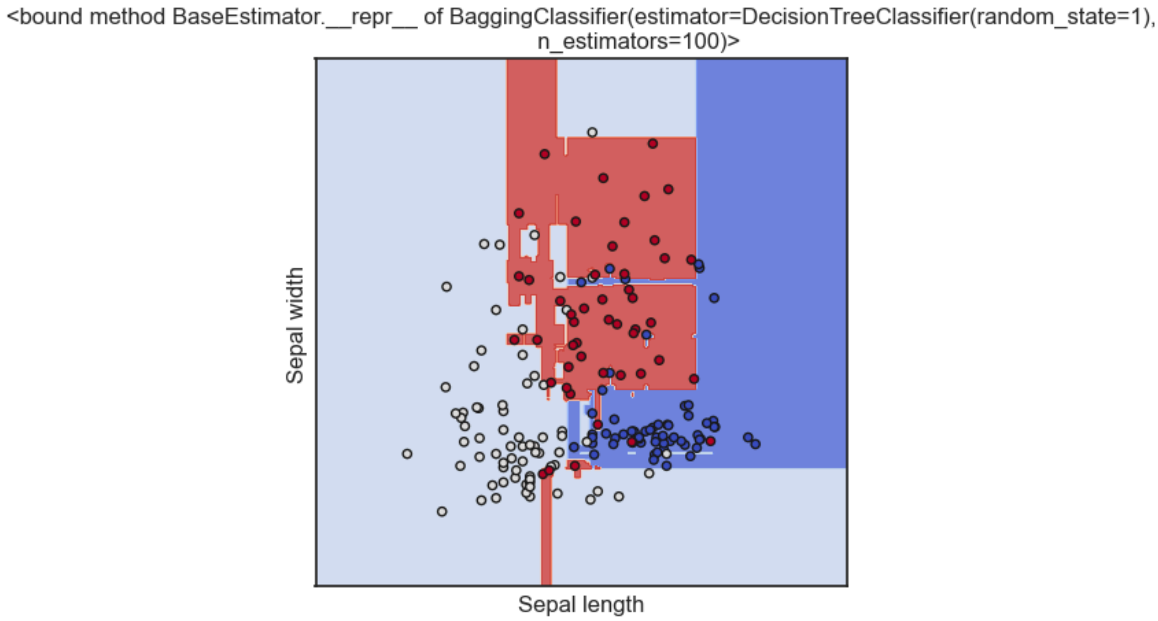
<bound method BaseEstimator.__repr__ of DecisionTreeClassifier(random_state=1)>




```
In [79]: plot_cl(BaggingClassifier(DecisionTreeClassifier(random_state=1), n_estimators=5
```

<bound method BaseEstimator.__repr__ of BaggingClassifier(estimator=DecisionTreeClassifier(random_state=1), n_estimators=5)>


```
In [80]: plot_cl(BaggingClassifier(DecisionTreeClassifier(random_state=1), n_estimators=1
```



Оценка качества модели бэггинга с помощью метрик accuracy и F-меры

In [81]:
```
cl1_2 = BaggingClassifier(n_estimators=5, oob_score=True, random_state=10)
cl1_2.fit(wine_X_train, wine_y_train)
target1_2 = cl1_2.predict(wine_X_test)
len(target1_2), target1_2
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\ensemble\_bagging.py:769: User
Warning: Some inputs do not have OOB scores. This probably means too few estimato
rs were used to compute any reliable oob estimates.
  warn(
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\ensemble\_bagging.py:775: Runt
imeWarning: invalid value encountered in divide
  oob_decision_function = predictions / predictions.sum(axis=1)[:, np.newaxis]
```

Out[81]:
```
(54,
 array([2, 1, 0, 1, 0, 2, 1, 0, 2, 1, 0, 0, 1, 0, 1, 1, 2, 0, 1, 0, 0, 1,
        2, 1, 0, 2, 0, 0, 0, 2, 1, 2, 2, 0, 1, 1, 1, 1, 1, 0, 0, 1, 2, 0,
        0, 0, 1, 0, 0, 0, 1, 2, 2, 0]))
```

In [82]:
```
accuracy_score(wine_y_test, target1_2)
```

Out[82]:
```
1.0
```

In [83]:
```
classification_report(wine_y_test, target1_2,
                      target_names=wine.target_names, output_dict=True)
```

Out[83]:
```
{'class_0': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 23.0},
 'class_1': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 19.0},
 'class_2': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 12.0},
 'accuracy': 1.0,
 'macro avg': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 54.0},
 'weighted avg': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 54.0}}
```

Вывод: модель, полученная с помощью бэггинга очень точна, так как предсказанное полностью совпало с ожидаемым.

2. Случайный лес

In [84]:
```
# Обучим классификатор на 5 деревьях
tree1 = RandomForestClassifier(n_estimators=5, oob_score=True, random_state=10)
tree1.fit(wine_X, wine_y)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:615: UserW
arning: Some inputs do not have OOB scores. This probably means too few trees wer
e used to compute any reliable OOB estimates.
  warn(
```

Out[84]:
```
RandomForestClassifier
RandomForestClassifier(n_estimators=5, oob_score=True, random_state=10)
```

In [85]:
```
# Out-of-bag error, возвращаемый классификатором
tree1.oob_score_, 1-tree1.oob_score_
```

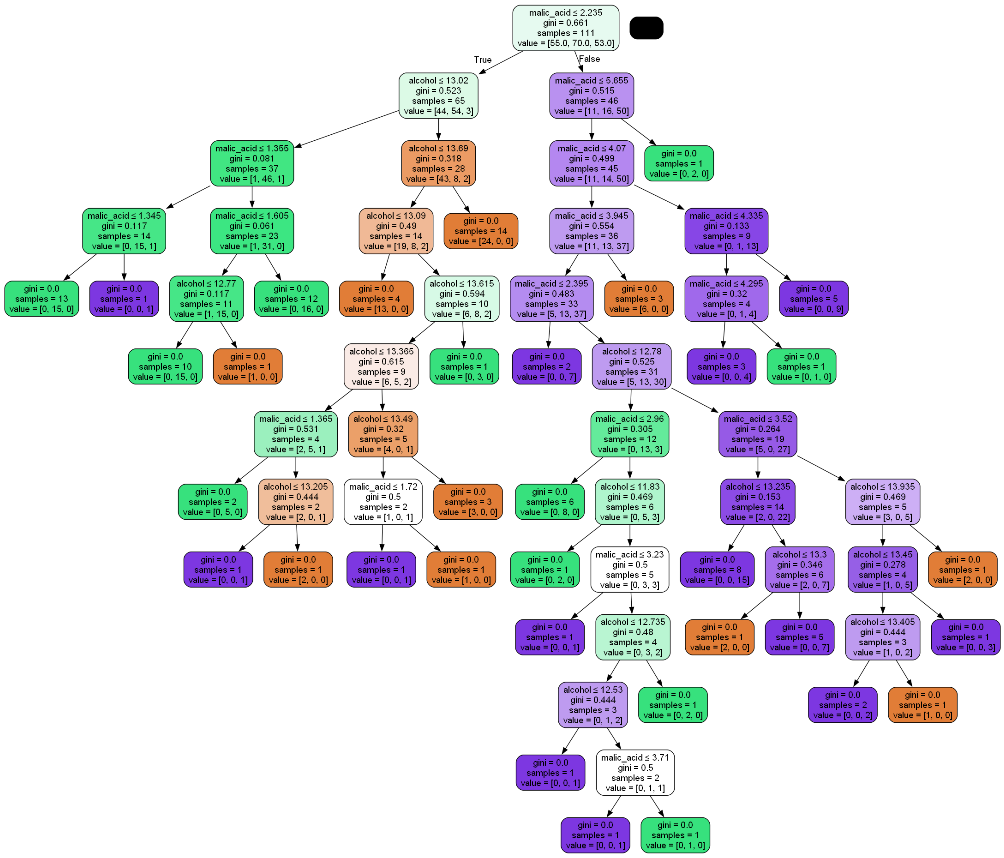
Out[85]:
```
(0.7134831460674157, 0.2865168539325843)
```

In [86]:
```
tree1.oob_decision_function_[55:70]
```

Out[86]:
```
array([[1.        , 0.        , 0.        ],
       [1.        , 0.        , 0.        ],
       [1.        , 0.        , 0.        ],
       [1.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.5       , 0.5       , 0.        ],
       [0.        , 0.66666667, 0.33333333],
       [0.        , 1.        , 0.        ],
       [0.        , 1.        , 0.        ],
       [0.        , 0.        , 0.        ],
       [0.        , 1.        , 0.        ]])
```

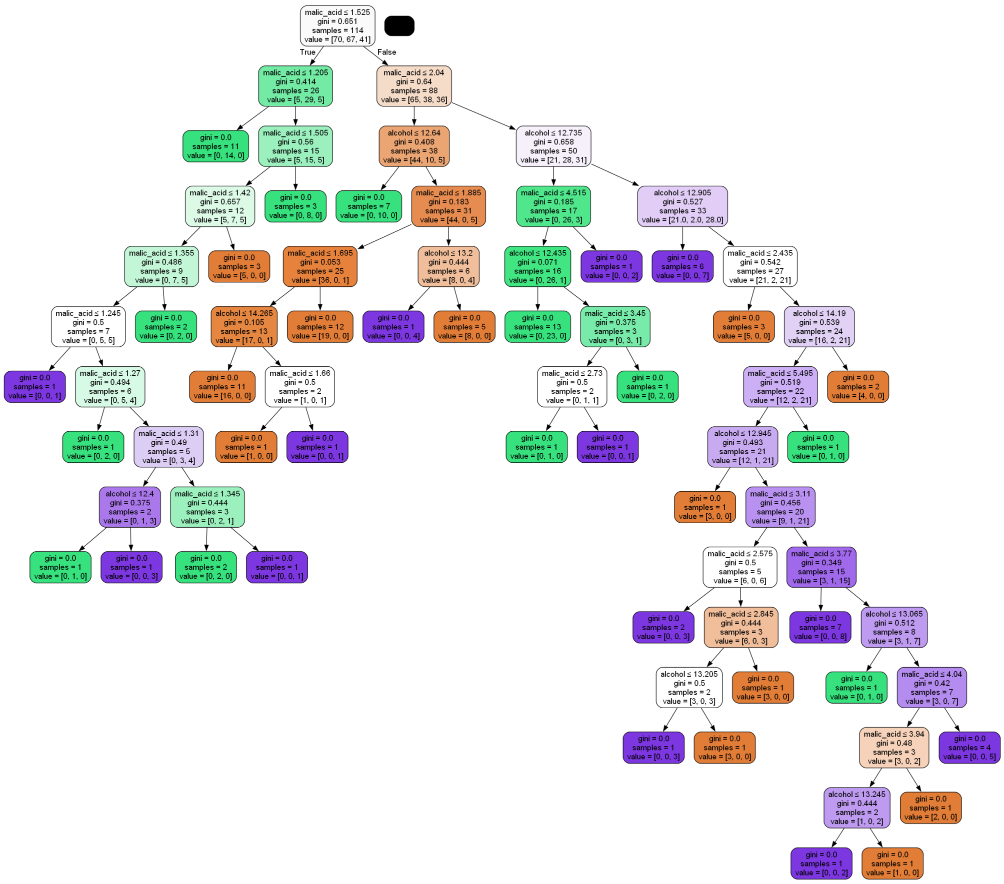
In [87]:
```
Image(get_png_tree(tree1.estimators_[0], wine.feature_names[:2]), width="500")
```

Out[87]:



In [88]:
```
Image(get_png_tree(tree1.estimators_[1], wine.feature_names[:2]), width="500")
```

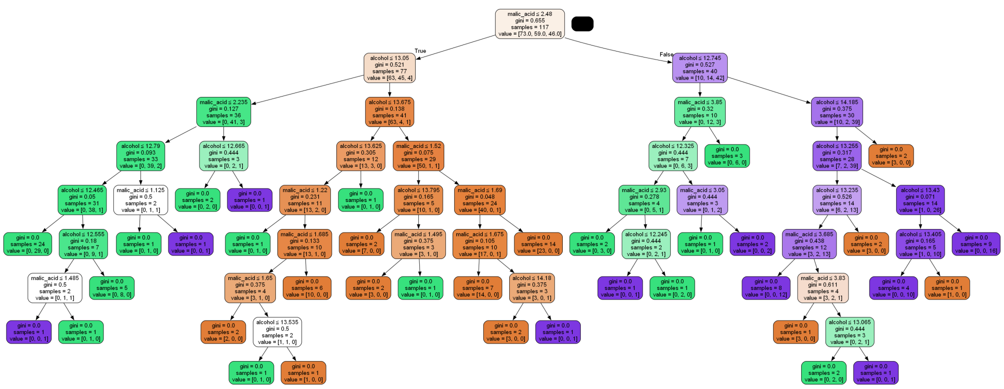
Out[88]:



In [89]:

```python
Image(get_png_tree(tree1.estimators_[2], wine.feature_names[:2]), width="500")
```

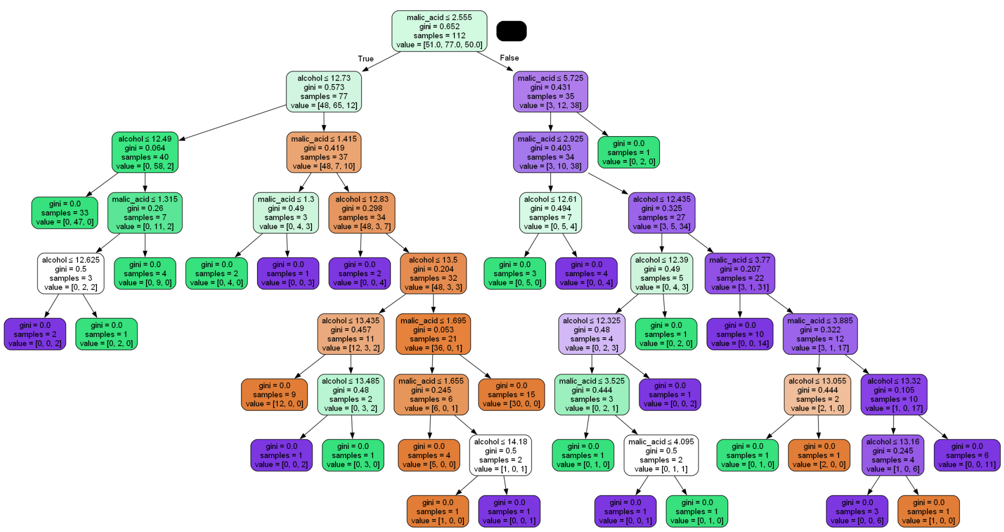
Out[89]:



In [90]:

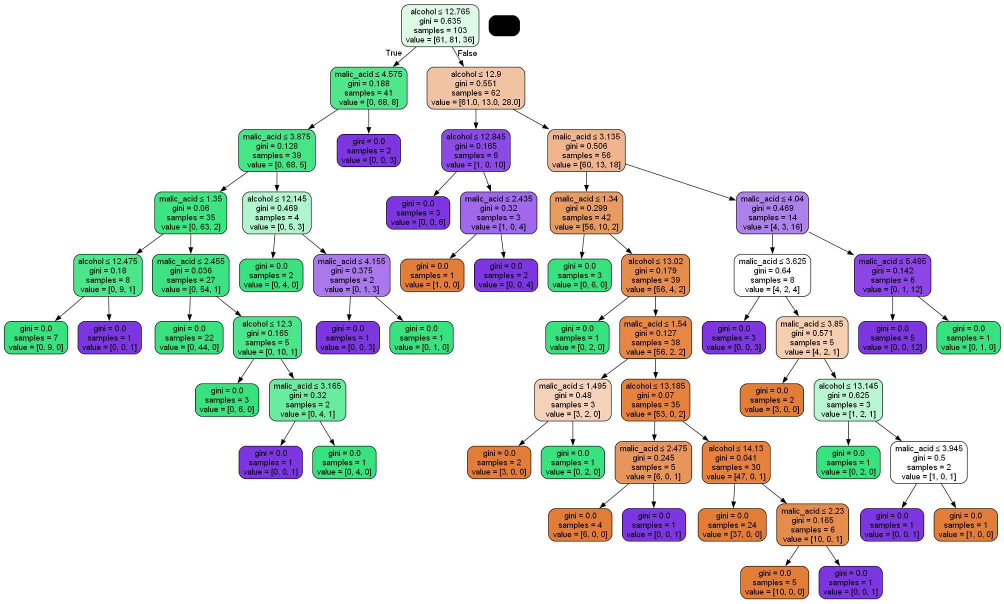
```python
Image(get_png_tree(tree1.estimators_[3], wine.feature_names[:2]), width="500")
```

Out[90]:



In [91]:
```
Image(get_png_tree(tree1.estimators_[4], wine.feature_names[:2]), width="500")
```

Out[91]:



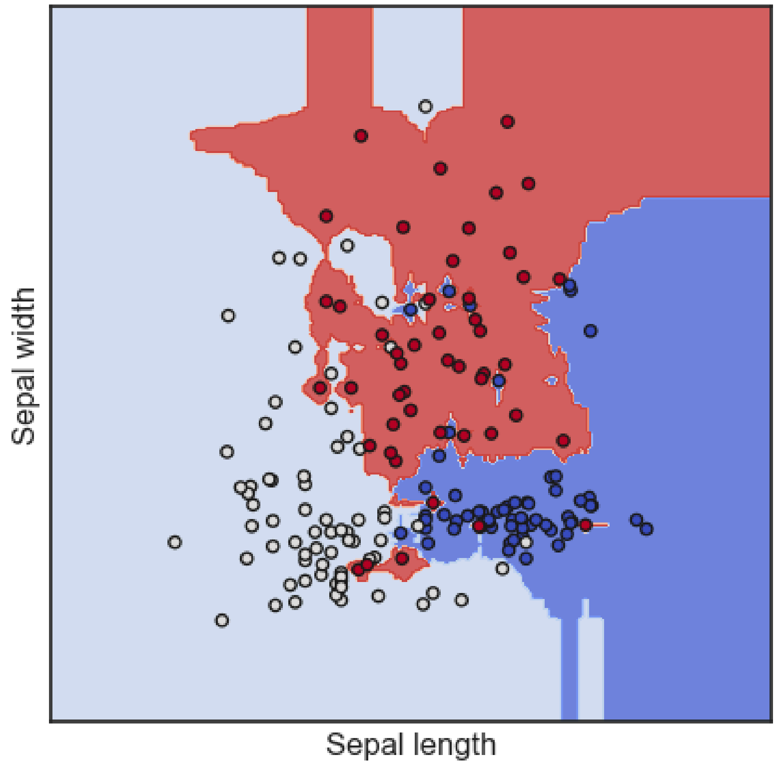
Визуализация результатов

In [92]:
```
plot_cl(RandomForestClassifier(random_state=1))
```

<bound method BaseEstimator.__repr__ of RandomForestClassifier(random_state=1)>

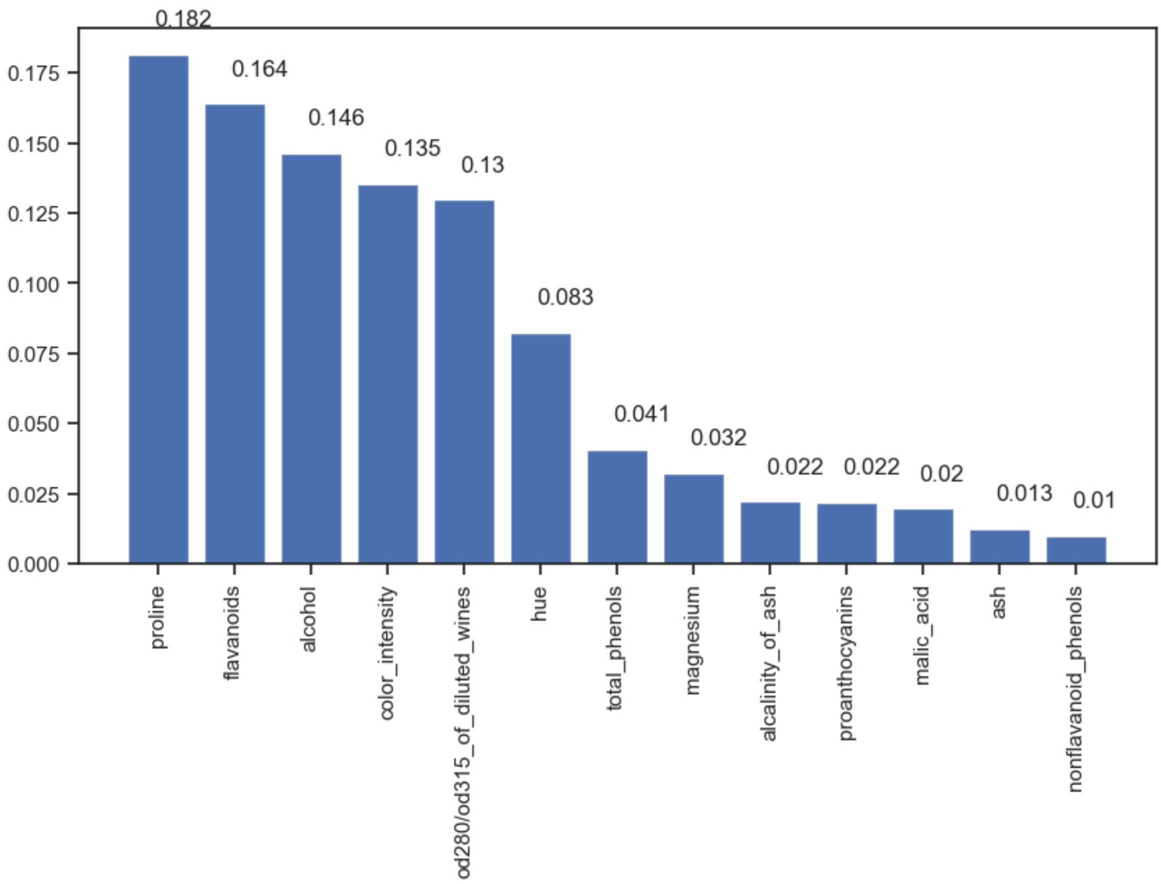
In [93]:
```
plot_cl(ExtraTreesClassifier(random_state=1))
```

<bound method BaseEstimator.__repr__ of ExtraTreesClassifier(random_state=1)>



In [94]:

```
# Важность признаков
wine_x_ds = pd.DataFrame(data=wine['data'], columns=wine['feature_names'])
wine_rf_cl = RandomForestClassifier(random_state=1)
wine_rf_cl.fit(wine_x_ds, wine.target)
_,_ = draw_feature_importances(wine_rf_cl, wine_x_ds)
```



Оценка качества модели случайный лес с помощью метрик accuracy и F-меры

In [95]:
```
cl1_2 = RandomForestClassifier(n_estimators=5, oob_score=True, random_state=10)
cl1_2.fit(wine_X_train, wine_y_train)
target1_2 = cl1_2.predict(wine_X_test)
len(target1_2), target1_2
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\ensemble\_forest.py:615: UserW
arning: Some inputs do not have OOB scores. This probably means too few trees wer
e used to compute any reliable OOB estimates.
  warn(
```

Out[95]:
```
(54,
 array([2, 1, 0, 1, 0, 2, 1, 0, 2, 1, 0, 0, 1, 0, 1, 1, 2, 0, 1, 0, 0, 1,
        2, 1, 0, 2, 0, 0, 0, 2, 1, 2, 2, 0, 1, 1, 1, 0, 1, 0, 0, 1, 2, 0,
        0, 0, 1, 0, 0, 0, 1, 2, 2, 0]))
```

In [96]:
```
accuracy_score(wine_y_test, target1_2)
```

Out[96]:
```
0.9814814814814815
```

In [97]:
```
classification_report(wine_y_test, target1_2,
                      target_names=wine.target_names, output_dict=True)
```

Out[97]:
```
{'class_0': {'precision': 0.9583333333333334,
  'recall': 1.0,
  'f1-score': 0.9787234042553191,
  'support': 23.0},
 'class_1': {'precision': 1.0,
  'recall': 0.9473684210526315,
  'f1-score': 0.972972972972973,
  'support': 19.0},
 'class_2': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 12.0},
 'accuracy': 0.9814814814814815,
 'macro avg': {'precision': 0.9861111111111112,
  'recall': 0.9824561403508771,
  'f1-score': 0.9838987924094308,
  'support': 54.0},
 'weighted avg': {'precision': 0.9822530864197532,
  'recall': 0.9814814814814815,
  'f1-score': 0.9814282367473858,
  'support': 54.0}}
```

Вывод: модель, полученная с помощью бэггинга весьма точна, так как предсказанное совпало с ожидаемым на 95-98%.

3. AdaBoost

In [98]:
```
# Обучим классификатор на 5 деревьях
ab1 = AdaBoostClassifier(n_estimators=5, algorithm='SAMME', random_state=10)
ab1.fit(wine_X, wine_y)
```

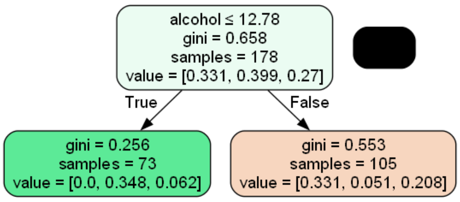
Out[98]:

AdaBoostClassifier

```
AdaBoostClassifier(algorithm='SAMME', n_estimators=5, random_state=10)
```

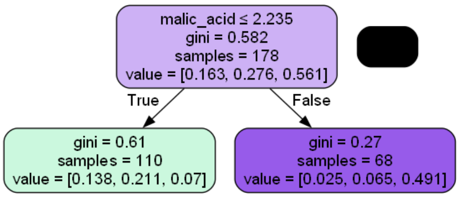
In [99]:
```
Image(get_png_tree(ab1.estimators_[0], wine.feature_names[:2]), width='40%')
```

Out[99]:



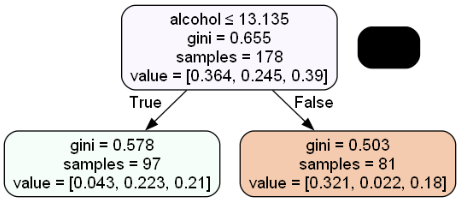
In [100…
```
Image(get_png_tree(ab1.estimators_[1], wine.feature_names[:2]), width='40%')
```

Out[100…



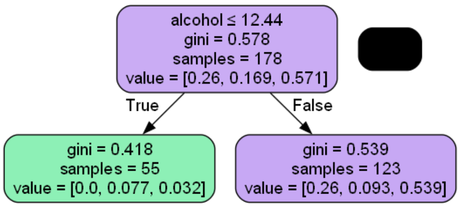
In [101…
```
Image(get_png_tree(ab1.estimators_[2], wine.feature_names[:2]), width='40%')
```

Out[101…



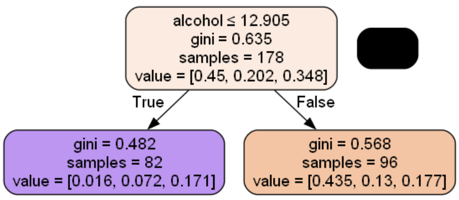
In [102…
```
Image(get_png_tree(ab1.estimators_[3], wine.feature_names[:2]), width='40%')
```

Out[102…



In [103…
```
Image(get_png_tree(ab1.estimators_[4], wine.feature_names[:2]), width='40%')
```

Out[103…



In [104…
```
ab1.estimator_weights_
```

Out[104…
```
array([1.44588646, 1.55274203, 0.87050387, 1.16201305, 1.12315087])
```

In [105…
```
df1 = ab1.decision_function(wine_X)
df1.shape
```

Out[105…
```
(178, 3)
```

In [106…
```
df1[:10]
```

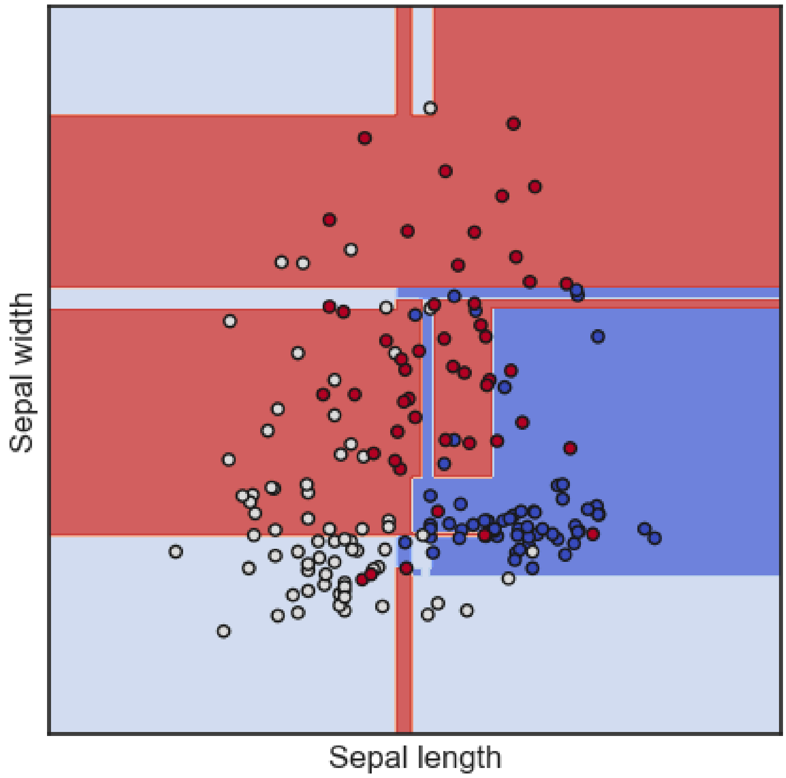
Out[106…
```
array([[ 0.33832685, -0.12154681, -0.21678004],
       [ 0.33832685, -0.12154681, -0.21678004],
       [ 0.33832685, -0.5       ,  0.16167315],
       [ 0.33832685, -0.12154681, -0.21678004],
       [ 0.33832685, -0.5       ,  0.16167315],
       [ 0.33832685, -0.12154681, -0.21678004],
       [ 0.33832685, -0.12154681, -0.21678004],
       [ 0.33832685, -0.12154681, -0.21678004],
       [ 0.33832685, -0.12154681, -0.21678004],
       [ 0.33832685, -0.12154681, -0.21678004]])
```

Визуализация результатов

In [107…
```python
# Результаты классификации
plot_cl(AdaBoostClassifier(random_state=1))
```

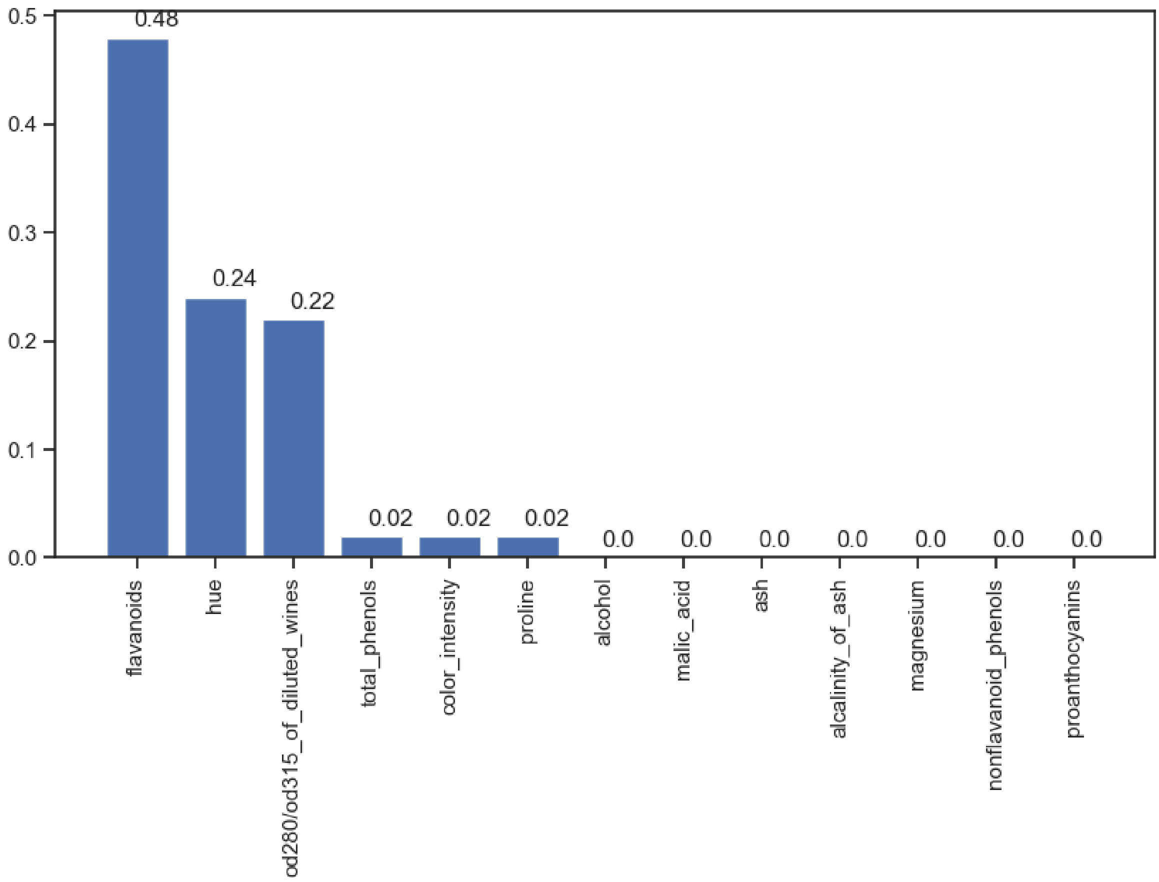
```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:5
19: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be
removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```

<bound method BaseEstimator.__repr__ of AdaBoostClassifier(random_state=1)>



In [108…
```python
# Важность признаков
wine_x_ds = pd.DataFrame(data=wine['data'], columns=wine['feature_names'])
ab2 = AdaBoostClassifier(random_state=1)
ab2.fit(wine_x_ds, wine.target)
_,_ = draw_feature_importances(ab2, wine_x_ds)
```

```
C:\ProgramData\anaconda3\Lib\site-packages\sklearn\ensemble\_weight_boosting.py:5
19: FutureWarning: The SAMME.R algorithm (the default) is deprecated and will be
removed in 1.6. Use the SAMME algorithm to circumvent this warning.
  warnings.warn(
```



Оценка качества модели AdaBoost с помощью метрик accuracy и F-меры

In [109…

```
cl1_2 = AdaBoostClassifier(n_estimators=5, algorithm='SAMME', random_state=10)
cl1_2.fit(wine_X_train, wine_y_train)
target1_2 = cl1_2.predict(wine_X_test)
len(target1_2), target1_2
```

Out[109…

```
(54,
 array([2, 1, 0, 1, 0, 2, 1, 0, 2, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1,
        1, 1, 0, 2, 0, 0, 0, 2, 1, 2, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 2, 0,
        0, 0, 1, 0, 0, 0, 1, 2, 2, 0]))
```

In [110…

```
accuracy_score(wine_y_test, target1_2)
```

Out[110…

```
0.9444444444444444
```

In [111…

```
classification_report(wine_y_test, target1_2,
                      target_names=wine.target_names, output_dict=True)
```

Out[111…

```
{'class_0': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 23.0},
 'class_1': {'precision': 0.8636363636363636,
  'recall': 1.0,
  'f1-score': 0.926829268292683,
  'support': 19.0},
 'class_2': {'precision': 1.0,
  'recall': 0.75,
  'f1-score': 0.8571428571428571,
  'support': 12.0},
 'accuracy': 0.9444444444444444,
 'macro avg': {'precision': 0.9545454545454546,
  'recall': 0.9166666666666666,
  'f1-score': 0.9279907084785134,
  'support': 54.0},
 'weighted avg': {'precision': 0.952020202020202,
  'recall': 0.9444444444444444,
  'f1-score': 0.9425087108013938,
  'support': 54.0}}
```

Вывод: модель, полученная с помощью бэггинга не так точна, как предыдущие, так как предсказанное совпало с ожидаемым на 86-95%.

4. Градиентный бустинг

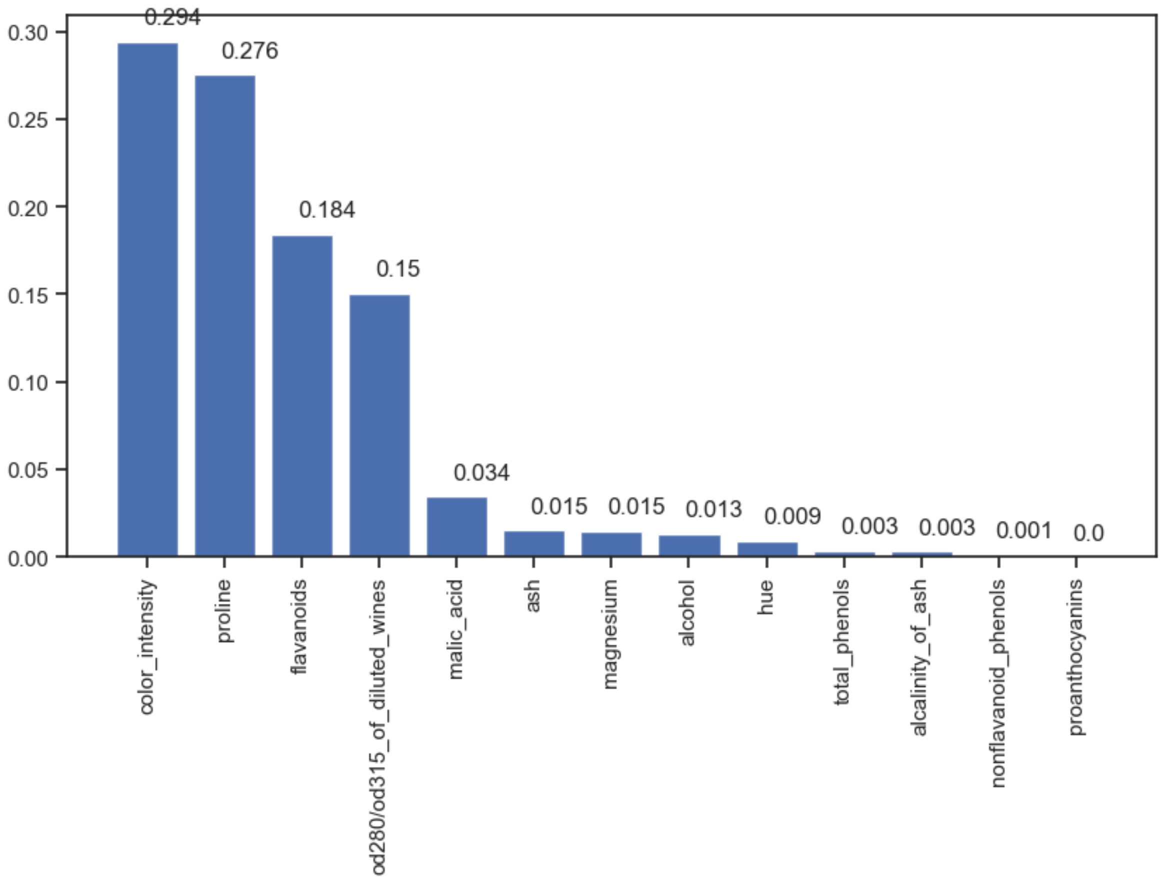
In [112…

```
plot_cl(GradientBoostingClassifier(random_state=1))
```

<bound method BaseEstimator.__repr__ of GradientBoostingClassifier(random_state=1)>

In [113…

```
# Важность признаков
wine_gb_cl = GradientBoostingClassifier(random_state=1)
wine_gb_cl.fit(wine_x_ds, wine.target)
_,_ = draw_feature_importances(wine_gb_cl, wine_x_ds)
```



In [114…

```
cl1_2 = GradientBoostingClassifier(random_state=1)
cl1_2.fit(wine_X_train, wine_y_train)
target1_2 = cl1_2.predict(wine_X_test)
len(target1_2), target1_2
```

Out[114…

```
(54,
 array([2, 1, 0, 1, 0, 2, 1, 0, 2, 1, 0, 0, 1, 0, 1, 1, 2, 0, 1, 0, 0, 1,
        2, 0, 0, 2, 0, 0, 0, 2, 1, 2, 2, 0, 1, 1, 1, 0, 1, 0, 0, 1, 2, 0,
        0, 0, 1, 0, 0, 0, 1, 2, 2, 0]))
```

In [115…

```
accuracy_score(wine_y_test, target1_2)
```

Out[115…

```
0.9629629629629629
```

In [116…

```
classification_report(wine_y_test, target1_2,
                      target_names=wine.target_names, output_dict=True)
```

Out[116…

```
{'class_0': {'precision': 0.92,
  'recall': 1.0,
  'f1-score': 0.9583333333333334,
  'support': 23.0},
 'class_1': {'precision': 1.0,
  'recall': 0.8947368421052632,
  'f1-score': 0.9444444444444444,
  'support': 19.0},
 'class_2': {'precision': 1.0,
  'recall': 1.0,
  'f1-score': 1.0,
  'support': 12.0},
 'accuracy': 0.9629629629629629,
 'macro avg': {'precision': 0.9733333333333333,
  'recall': 0.9649122807017544,
  'f1-score': 0.9675925925925926,
  'support': 54.0},
 'weighted avg': {'precision': 0.9659259259259259,
  'recall': 0.9629629629629629,
  'f1-score': 0.9627057613168725,
  'support': 54.0}}
```

Вывод: модель, полученная с помощью градиентного бустинга весьма точна, так как предсказанное совпало с ожидаемым на 94-96%.