Ларионова А.П. ИУ5-63Б Лабораторная работа №4

In [ ]:
```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from scipy.optimize import fmin_tnc
from IPython.display import Image
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_rep
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_e
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.metrics import plot_confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

Выберем датасет wine для задач классификации.

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

In [ ]:
```
df_wine = pd.DataFrame(wine.data,columns=wine.feature_names)
df_wine['target'] = pd.Series(wine.target)
df_wine.head()
```

Out[ ]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phen
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0

In [ ]:
```
wine.target_names
```

Out[ ]:
```
array(['class_0', 'class_1', 'class_2'], dtype='<U7')
```

In [ ]:
```
# Разделение выборки на обучающую и тестовую
```

```
wine_X_train, wine_X_test, wine_y_train, wine_y_test = train_test_split(
    wine.data, wine.target, test_size=0.5, random_state=1)
```

Логистическая регрессия для решения задач классификации.

In [ ]:
```
cl1 = LogisticRegression()
```

In [ ]:
```
cl1.fit(wine_X_train, wine_y_train)
```

```
c:\Users\amina\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:763: Conver
genceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[ ]:
```
LogisticRegression()
```

In [ ]:
```
pred_wine_y_test = cl1.predict(wine_X_test)
pred_wine_y_test
```

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

In [ ]:
```
pred_wine_y_test_proba = cl1.predict_proba(wine_X_test)
pred_wine_y_test_proba[:10]
```

Out[ ]:
```
array([[8.82902064e-03, 8.15199065e-03, 9.83018989e-01],
       [5.24655698e-05, 9.99932780e-01, 1.47543278e-05],
       [9.72089096e-01, 1.96478986e-02, 8.26300508e-03],
       [3.20488782e-01, 6.77041180e-01, 2.47003782e-03],
       [9.97950734e-01, 5.71075989e-06, 2.04355535e-03],
       [3.69505272e-03, 1.20332280e-02, 9.84271719e-01],
       [1.61841069e-01, 7.17264002e-01, 1.20894929e-01],
       [9.99919597e-01, 3.41985078e-08, 8.03687620e-05],
       [2.19405370e-04, 2.67520734e-04, 9.99513074e-01],
       [1.69949365e-03, 9.94032729e-01, 4.26777742e-03]])
```

In [ ]:
```
# Вероятность принадлежности к 0 классу
[round(x, 4) for x in pred_wine_y_test_proba[:10,0]]
```

Out[ ]:
```
[0.0088, 0.0001, 0.9721, 0.3205, 0.998, 0.0037, 0.1618, 0.9999, 0.0002, 0.0017]
```

In [ ]:
```
# Вероятность принадлежности к 1 классу
[round(x, 4) for x in pred_wine_y_test_proba[:10,1]]
```

Out[ ]:
```
[0.0082, 0.9999, 0.0196, 0.677, 0.0, 0.012, 0.7173, 0.0, 0.0003, 0.994]
```

In [ ]:

```
# Вероятность принадлежности ко 2 классу
[round(x, 4) for x in pred_wine_y_test_proba[:10,2]]
```

Out[ ]:

```
[0.983, 0.0, 0.0083, 0.0025, 0.002, 0.9843, 0.1209, 0.0001, 0.9995, 0.0043]
```

In [ ]:

```
# Сумма вероятностей равна 1
pred_wine_y_test_proba[:10,0] + pred_wine_y_test_proba[:10,1]+pred_wine_y_test_proba[:1
```

Out[ ]:

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Оценим качество модели с помощью метрики Accuracy.

In [ ]:

```
accuracy_score(wine_y_test, pred_wine_y_test)
```

Out[ ]:

```
0.9438202247191011
```

In [ ]:

```
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]))
```

In [ ]:

```
print_accuracy_score_for_classes(wine_y_test, pred_wine_y_test)
```

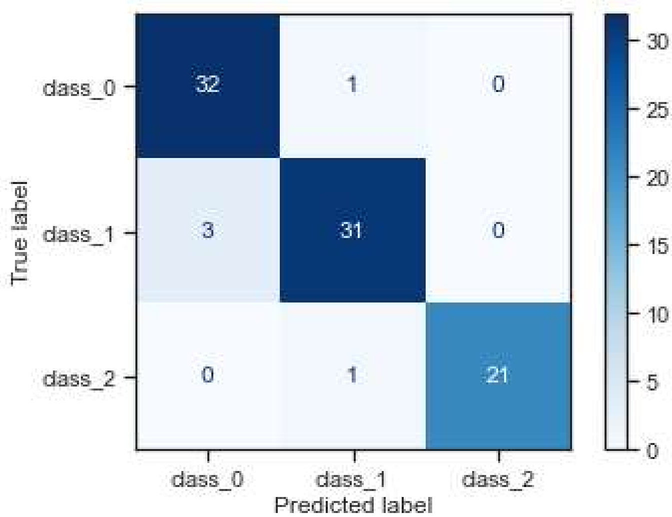
```
Метка    Accuracy
0        0.9696969696969697
1        0.9117647058823529
2        0.9545454545454546
```

Теперь оценим качество модели с помощью метрики Confusion Matrix.

In [ ]:

```
plot_confusion_matrix(cl1, wine_X_test, wine_y_test,
                      display_labels=wine.target_names, cmap=plt.cm.Blues)
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23f48694760>



Метод опорных векторов SVM.

In [ ]:

```
# возьмем два первых признака из нашего датасета
wine_X = wine.data[:, :2]
wine_y = wine.target
```

In [ ]:

```
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
```

In [ ]:

```
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
```

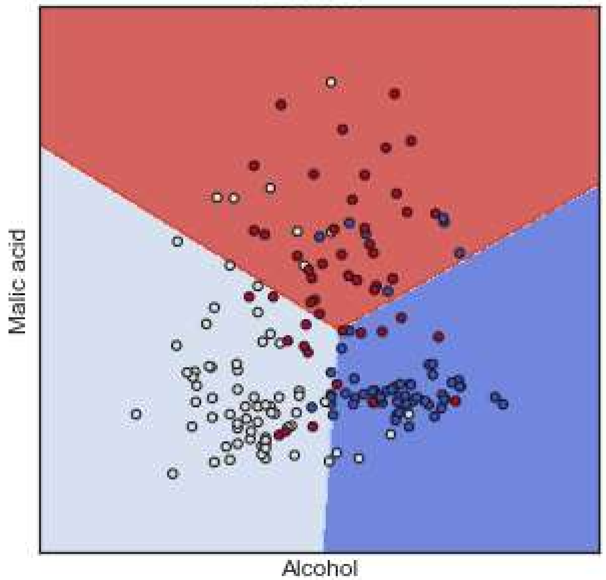
In [ ]:

```
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('Alcohol')
    ax.set_ylabel('Malic acid')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
    plt.show()
```

In [ ]:

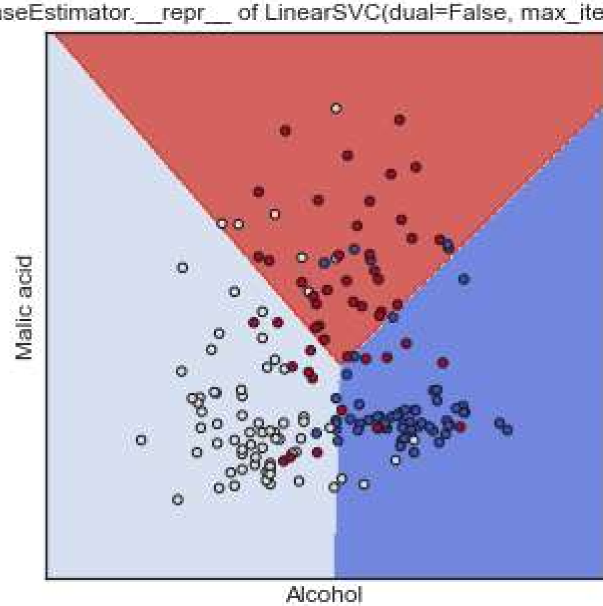
```
plot_cl(LinearSVC(C=1.0, max_iter=10000))
```

<bound method BaseEstimator.__repr__ of LinearSVC(max_iter=10000)>



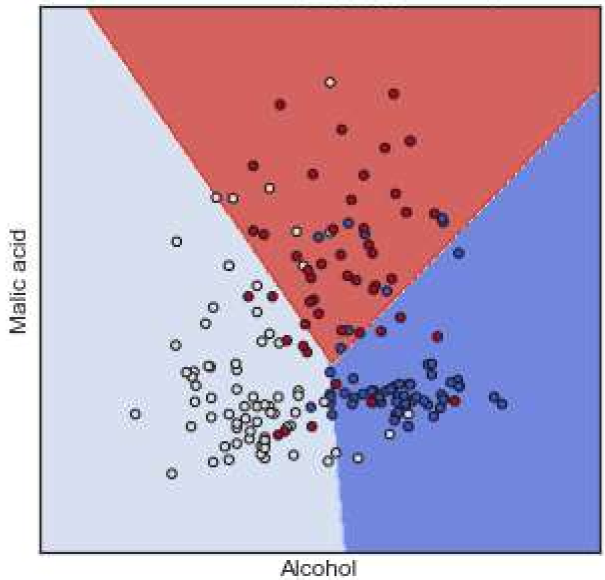
In [ ]:
```
plot_cl(LinearSVC(C=1.0, penalty='l1', dual=False, max_iter=10000))
```

<bound method BaseEstimator.__repr__ of LinearSVC(dual=False, max_iter=10000, penalty='l1')>



In [ ]:
```
plot_cl(SVC(kernel='linear', C=1.0))
```

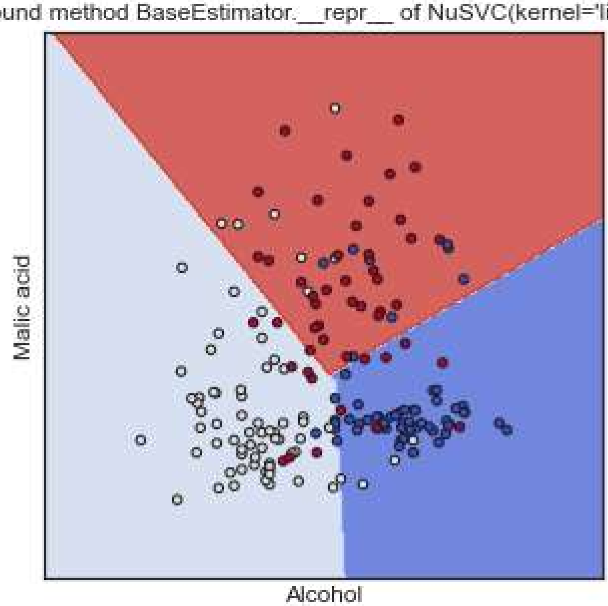
<bound method BaseEstimator.__repr__ of SVC(kernel='linear')>



In [ ]:

```
plot_cl(NuSVC(kernel='linear', nu=0.5))
```

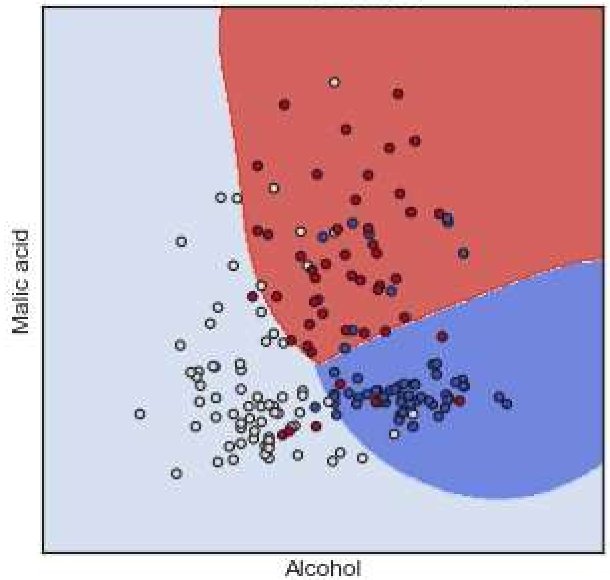
<bound method BaseEstimator.__repr__ of NuSVC(kernel='linear')>



In [ ]:

```
plot_cl(SVC(kernel='rbf', gamma=0.2, C=1.0))
```

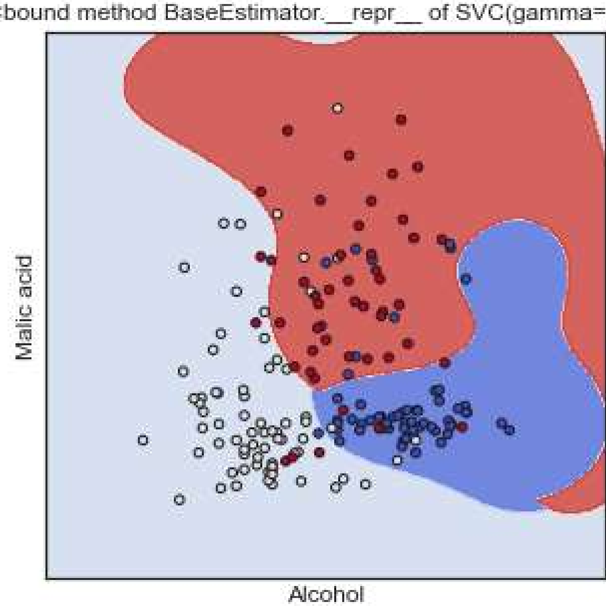
<bound method BaseEstimator.__repr__ of SVC(gamma=0.2)>



In [ ]:

```
plot_cl(SVC(kernel='rbf', gamma=0.9, C=1.0))
```

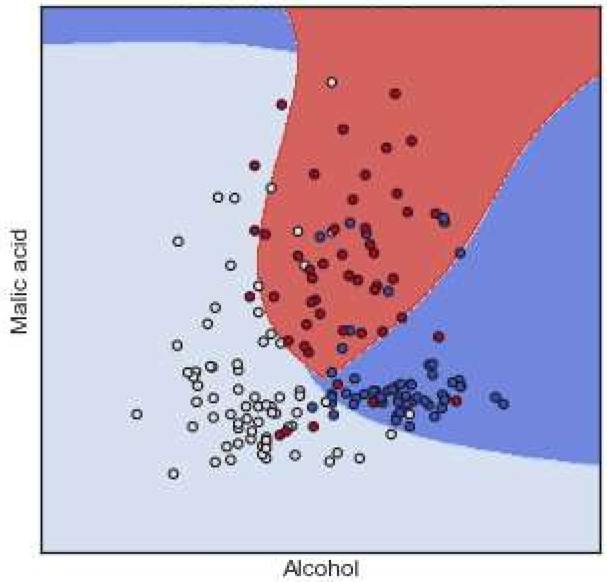
<bound method BaseEstimator.__repr__ of SVC(gamma=0.9)>



In [ ]:

```
plot_cl(SVC(kernel='poly', degree=4, gamma=0.2, C=1.0))
```

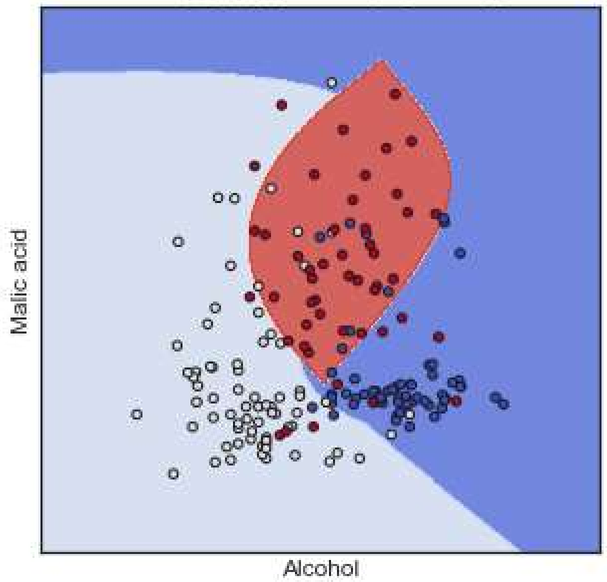
<bound method BaseEstimator.__repr__ of SVC(degree=4, gamma=0.2, kernel='poly')>



In [ ]:

```
plot_cl(SVC(kernel='poly', degree=4, gamma=0.9, C=1.0))
```

<bound method BaseEstimator.__repr__ of SVC(degree=4, gamma=0.9, kernel='poly')>



In [ ]:

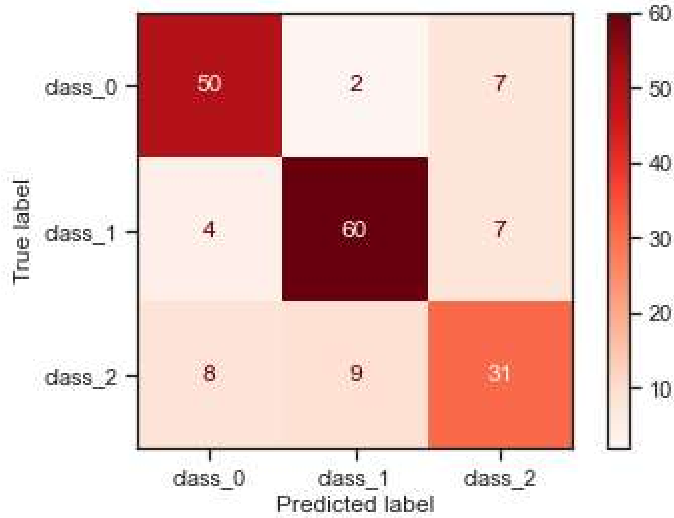
```
cl2=SVC()
cl2.fit(wine_X,wine_y)
pred_wine_y_test=cl2.predict(wine_X)
accuracy_score(wine_y, pred_wine_y_test)
```

Out[ ]: 0.7921348314606742

In [ ]:

```
plot_confusion_matrix(cl2, wine_X, wine_y,
                      display_labels=wine.target_names, cmap=plt.cm.Reds)
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23f49ae5bb0>

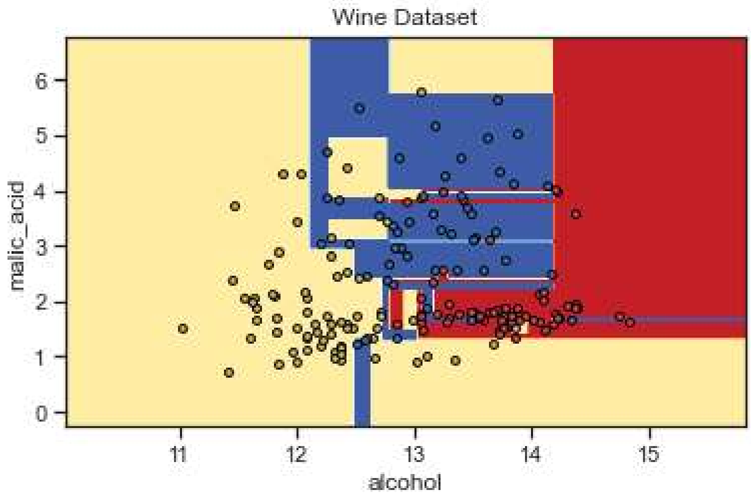


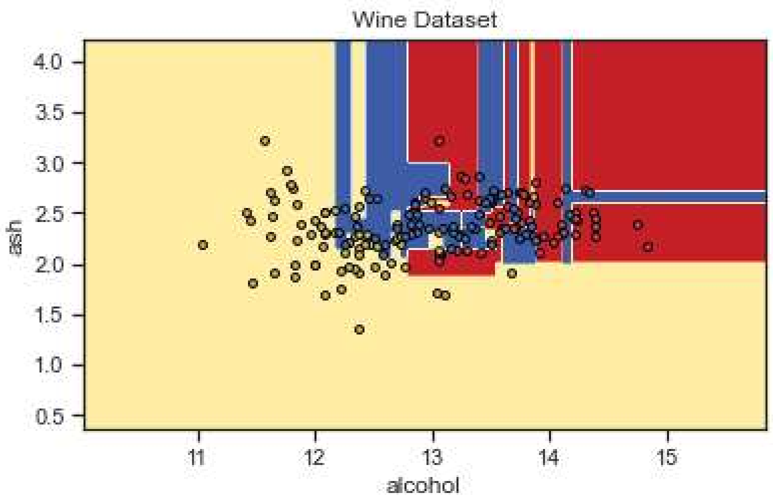
Дерево решений.

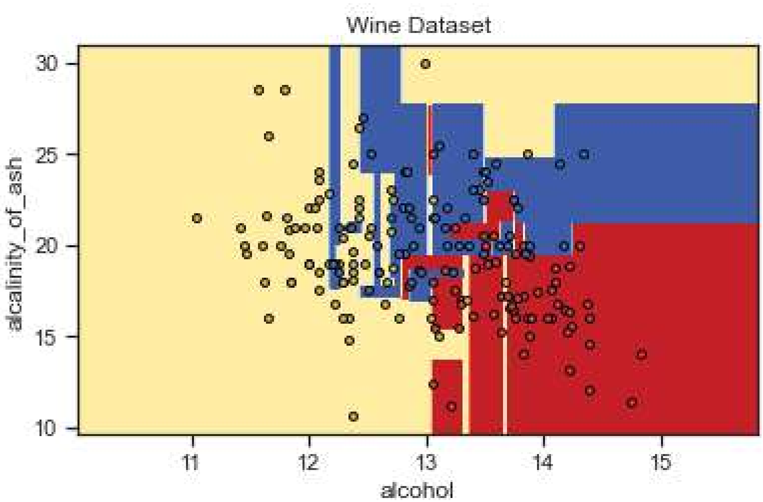
```
In [ ]:
def plot_tree_classification(title_param, ds):
    """
    Построение деревьев и вывод графиков для заданного датасета
    """

    n_classes = len(np.unique(ds.target))
    plot_colors = "ryb"
    plot_step = 0.02

    for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                    [1, 2], [1, 3], [2, 3]]):
        # We only take the two corresponding features
        X = ds.data[:, pair]
        y = ds.target

        # Train
        clf = DecisionTreeClassifier(random_state=1).fit(X, y)

        plt.title(title_param)

        x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
        y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
        xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                             np.arange(y_min, y_max, plot_step))
        plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)

        Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
        Z = Z.reshape(xx.shape)
        cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

        plt.xlabel(ds.feature_names[pair[0]])
        plt.ylabel(ds.feature_names[pair[1]])

        # Plot the training points
        for i, color in zip(range(n_classes), plot_colors):
            idx = np.where(y == i)
            plt.scatter(X[idx, 0], X[idx, 1], c=color, label=ds.target_names[i],
                        cmap=plt.cm.RdYlBu, edgecolor='black', s=15)

        plt.show()
```

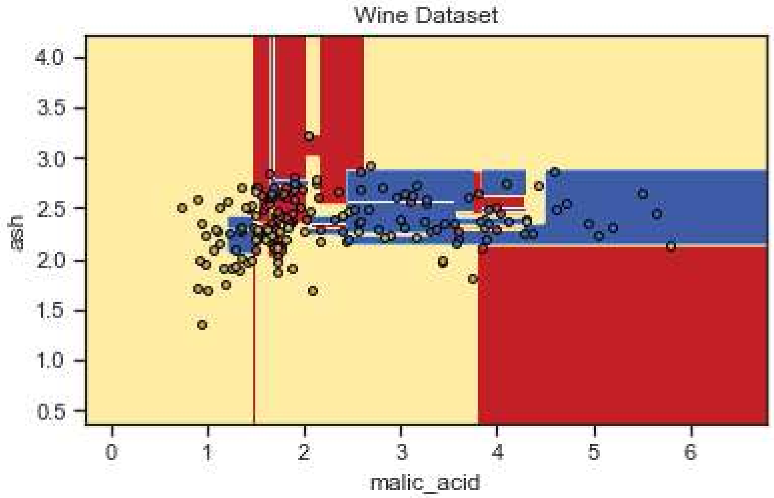
In [ ]:

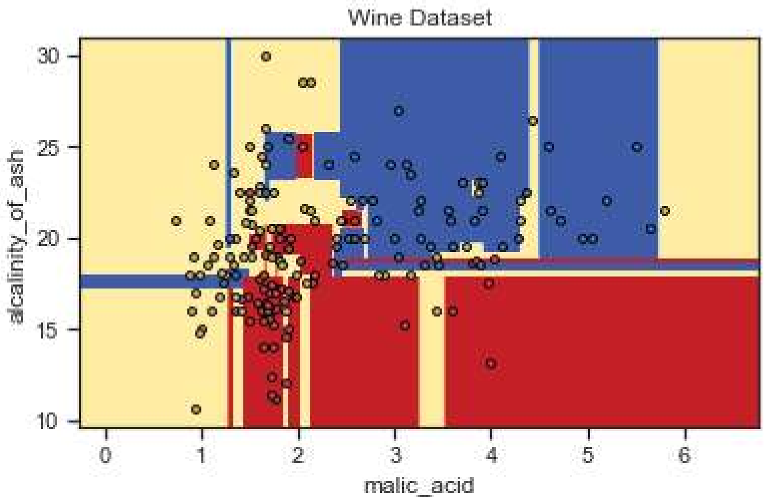
```
plot_tree_classification('Wine Dataset', wine)
```

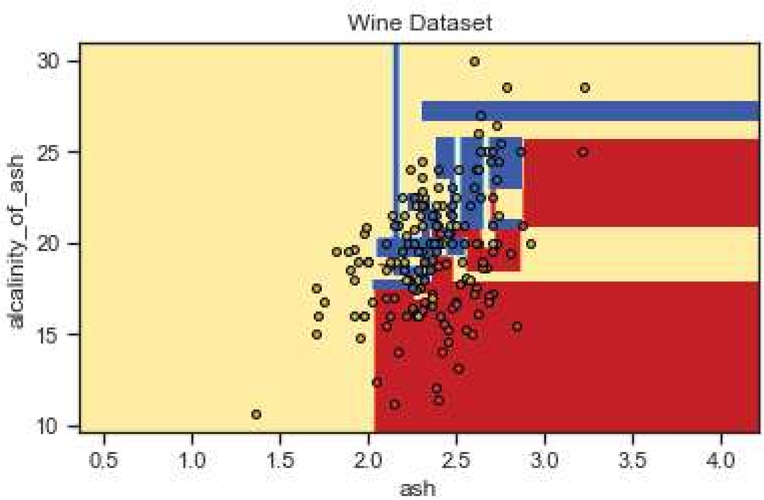












In [ ]:

```
cl3=DecisionTreeClassifier()
cl3.fit(wine_X_train,wine_y_train)
pred_wine_y_test=cl3.predict(wine_X_test)
accuracy_score(wine_y_test, pred_wine_y_test)
```

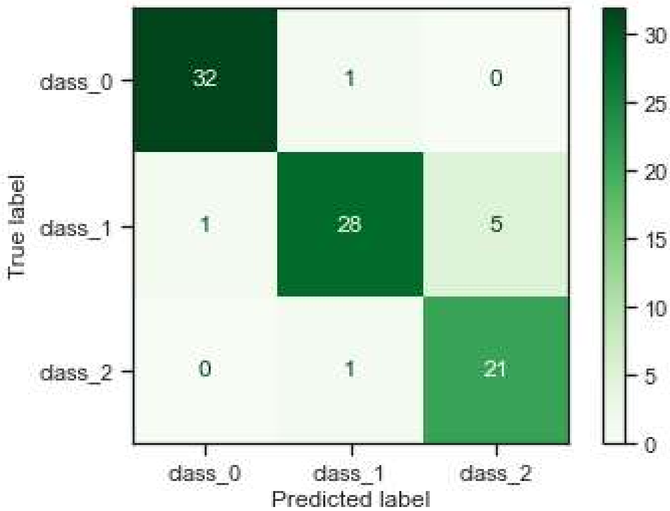
Out[ ]:

```
0.9101123595505618
```

In [ ]:

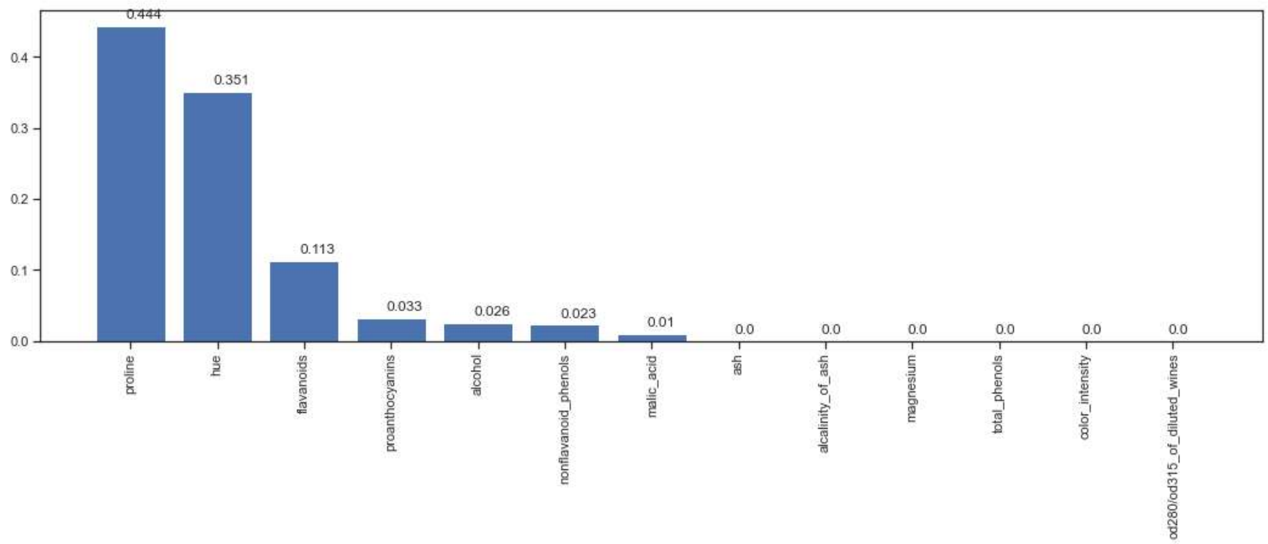
```
plot_confusion_matrix(cl3, wine_X_test, wine_y_test,
                      display_labels=wine.target_names, cmap=plt.cm.Greens)
```

Out[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x23f4af68280>



In [ ]:

```
from operator import itemgetter

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

draw_feature_importances(cl3, df_wine)
```



Out[ ]:

```
(['proline',
  'hue',
  'flavanoids',
  'proanthocyanins',
  'alcohol',
  'nonflavanoid_phenols',
  'malic_acid',
  'ash',
  'alcalinity_of_ash',
  'magnesium',
  'total_phenols',
  'color_intensity',
  'od280/od315_of_diluted_wines'],
 [0.4440366755963771,
  0.35099513076151634,
  0.11327272727272722,
  0.032742474916387966,
  0.025673076923076923,
  0.02282051282051282,
  0.010459401709401722,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0,
  0.0])
```

Вывод правил дерева в текстовом виде.

In [ ]:

```
from IPython.core.display import HTML
from sklearn.tree import export_text
tree_rules = export_text(cl3, feature_names=list(wine.feature_names))
HTML('<pre>' + tree_rules + '</pre>')
```

Out[ ]:

```
|--- proline <= 938.00
|   |--- hue <= 0.78
|   |   |--- proanthocyanins <= 3.14
|   |   |   |--- class: 2
|   |   |--- proanthocyanins >  3.14
|   |   |   |--- class: 1
|   |--- hue >  0.78
```

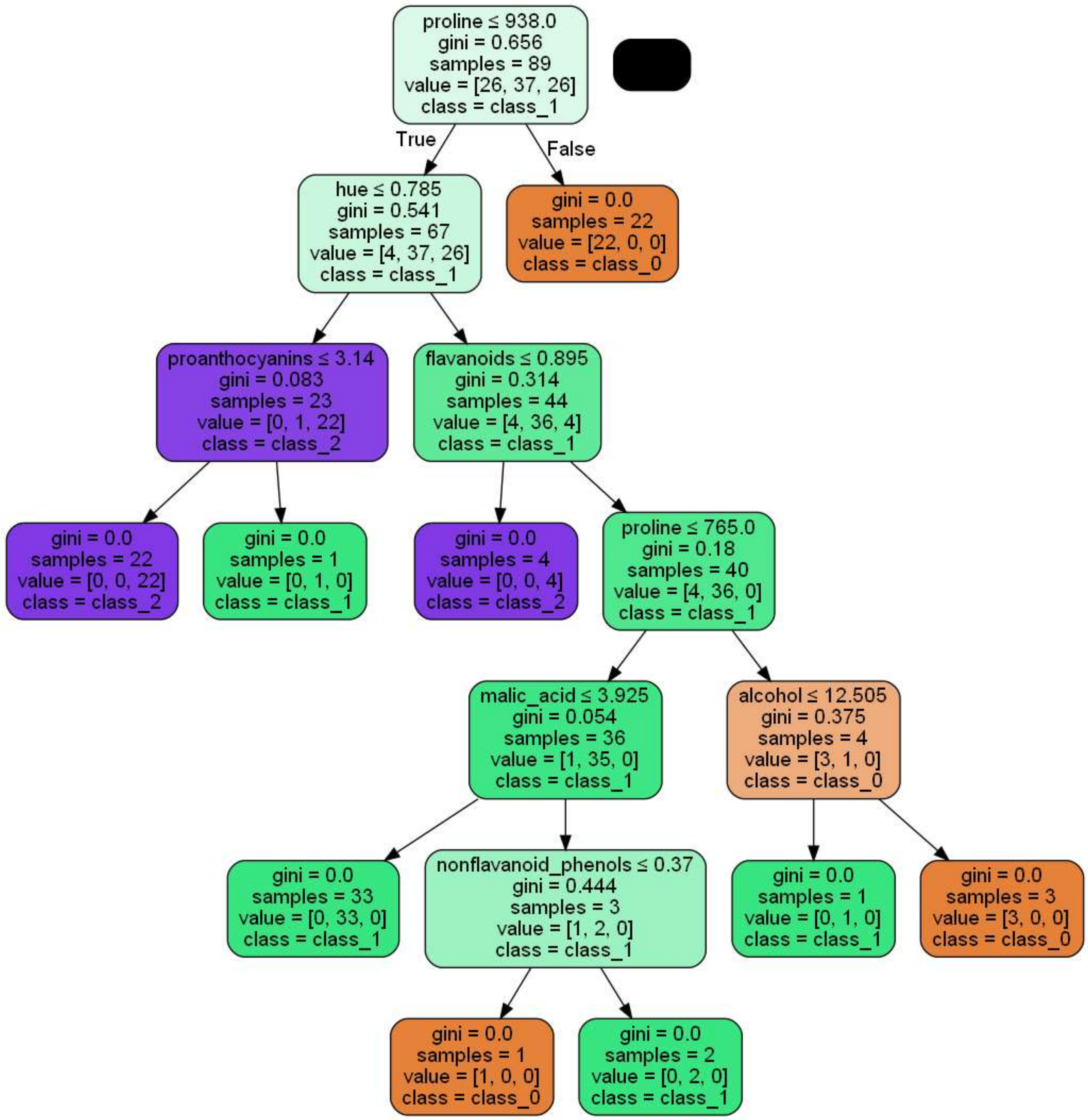
```
|   |   |--- flavanoids <= 0.90
|   |   |   |--- class: 2
|   |   |--- flavanoids >  0.90
|   |   |   |--- proline <= 765.00
|   |   |   |   |--- malic_acid <= 3.92
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- malic_acid >  3.92
|   |   |   |   |   |--- nonflavanoid_phenols <= 0.37
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- nonflavanoid_phenols >  0.37
|   |   |   |   |   |   |--- class: 1
|   |   |   |--- proline >  765.00
|   |   |   |   |--- alcohol <= 12.51
|   |   |   |   |   |--- class: 1
|   |   |   |   |--- alcohol >  12.51
|   |   |   |   |   |--- class: 0
|--- proline >  938.00
|   |--- class: 0
```

Визуализация дерева.

In [ ]:

```
from io import StringIO
from sklearn.tree import export_graphviz
import pydotplus

def get_png_tree(tree_model_param, feature_names_param, class_names):
    dot_data = StringIO()
    export_graphviz(tree_model_param, out_file=dot_data, feature_names=feature_names_pa
                    filled=True, rounded=True, special_characters=True, class_names=cla
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph.create_png()

Image(get_png_tree(cl3, wine.feature_names, wine.target_names), height='70%')
```

Out[ ]:


proline ≤ 938.0
gini = 0.656
samples = 89
value = [26, 37, 26]
class = class_1
True
False
hue ≤ 0.785
gini = 0.541
samples = 67
value = [4, 37, 26]
class = class_1
gini = 0.0
samples = 22
value = [22, 0, 0]
class = class_0
proanthocyanins ≤ 3.14
gini = 0.083
samples = 23
value = [0, 1, 22]
class = class_2
flavanoids ≤ 0.895
gini = 0.314
samples = 44
value = [4, 36, 4]
class = class_1
gini = 0.0
samples = 22
value = [0, 0, 22]
class = class_2
gini = 0.0
samples = 1
value = [0, 1, 0]
class = class_1
gini = 0.0
samples = 4
value = [0, 0, 4]
class = class_2
proline ≤ 765.0
gini = 0.18
samples = 40
value = [4, 36, 0]
class = class_1
malic_acid ≤ 3.925
gini = 0.054
samples = 36
value = [1, 35, 0]
class = class_1
alcohol ≤ 12.505
gini = 0.375
samples = 4
value = [3, 1, 0]
class = class_0
gini = 0.0
samples = 33
value = [0, 33, 0]
class = class_1
nonflavanoid_phenols ≤ 0.37
gini = 0.444
samples = 3
value = [1, 2, 0]
class = class_1
gini = 0.0
samples = 1
value = [0, 1, 0]
class = class_1
gini = 0.0
samples = 3
value = [3, 0, 0]
class = class_0
gini = 0.0
samples = 1
value = [1, 0, 0]
class = class_0
gini = 0.0
samples = 2
value = [0, 2, 0]
class = class_1