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

Отчет по лабораторной работе № 5

«Линейные модели, SVM и деревья решений»

по курсу
“Методы машинного обучения”

Выполнил:
Али Диб А.Ж.
Студент группы ИУ5-22М

Москва, 2020

Цель лабораторной работы:

изучение линейных моделей, SVM и деревьев решений.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from typing import Dict, Tuple
from sklearn.datasets import *
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
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
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_l
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_grap
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
from sklearn.linear_model import RidgeClassifier
%matplotlib inline
sns.set(style="ticks")
import warnings
warnings.filterwarnings("ignore")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: Future
  import pandas.util.testing as tm
```

Набор данных

В качестве набора данных мы будем использовать набор данных для выявления рака моло
https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)

```
cancer = load_breast_cancer()
data=pd.DataFrame(data=np.c_[cancer['data'],cancer['target']],
                  columns = list(cancer['feature_names']) + ['target'])
```

```
data.head()
```

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mea concav point
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.1471
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.0701
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.1279
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.1052
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.1043

```
data.shape
```

```
(569, 31)
```

```
data.columns
```

```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension',
       'target'],
      dtype='object')
```

```
data.dtypes
```

```
mean radius                float64
mean texture               float64
mean perimeter             float64
mean area                  float64
mean smoothness            float64
mean compactness           float64
mean concavity             float64
```

```
data.isnull().sum()
```

```
mean radius                0
mean texture               0
mean perimeter             0
mean area                  0
mean smoothness            0
mean compactness           0
mean concavity             0
mean concave points        0
mean symmetry              0
mean fractal dimension     0
radius error               0
texture error              0
perimeter error            0
area error                 0
smoothness error           0
compactness error          0
concavity error            0
concave points error       0
symmetry error             0
fractal dimension error    0
worst radius               0
worst texture              0
worst perimeter            0
worst area                 0
worst smoothness           0
worst compactness          0
worst concavity            0
worst concave points       0
worst symmetry             0
worst fractal dimension    0
target                     0
dtype: int64
```

База данных не содержит отсутствующих переменных или переменных категории. Мы мож

```
np.unique(data.target)
```

```
array([0., 1.])
```

Классы классификации состоят только из двух значений (0 или 1)

Разделите выборку на обучающую и тестовую

```
X_train, X_test, y_train, y_test = train_test_split(data[['mean compactness','radiu
```

```
'worst smoothness','wors
```

```
X_train.shape, y_train.shape
```

```
((113, 9), (113,))
```

```
X_test.shape, y_test.shape
```

```
((456, 9), (456,))
```

```
np.unique(y_train)
```

```
array([0., 1.])
```

```
np.unique(y_test)
```

```
array([0., 1.])
```

Обучение

Сначала давайте посмотрим, сколько значений в каждом классе.

```
def class_proportions(array: np.ndarray) -> Dict[int, Tuple[int, float]]:
    """
    Вычисляет пропорции классов
    array - массив, содержащий метки классов
    """
    # Получение меток классов и количества меток каждого класса
    labels, counts = np.unique(array, return_counts=True)
    # Превращаем количество меток в процент их встречаемости
    # делим количество меток каждого класса на общее количество меток
    counts_perc = counts/array.size
    # Теперь sum(counts_perc)==1.0
    # Создаем результирующий словарь,
    # ключом словаря является метка класса,
    # а значением словаря процент встречаемости метки
    res = dict()
    for label, count2 in zip(labels, zip(counts, counts_perc)):
        res[label] = count2
    return res

def print_class_proportions(array: np.ndarray):
    """
    Вывод пропорций классов
    """
    proportions = class_proportions(array)
    if len(proportions)>0:
        print('Метка \t Количество \t Процент встречаемости')
    for i in proportions:
```

```
        val, val_perc = proportions[i]
        val_perc_100 = round(val_perc * 100, 2)
        print('{} \t {} \t \t {}%'.format(i, val, val_perc_100))


print_class_proportions(data.target)
```

```
Метка   Количество      Процент встречаемости
0.0     212             37.26%
1.0     357             62.74%
```

Таким образом, в наборе данных есть небольшой уклон (bias)

Оценка качества моделей

для оценки качества каждого классификатора. Мы будем использовать precision, recall, f1 и
В дополнение к этому мы увидим оценку точности для двух классов отдельно.

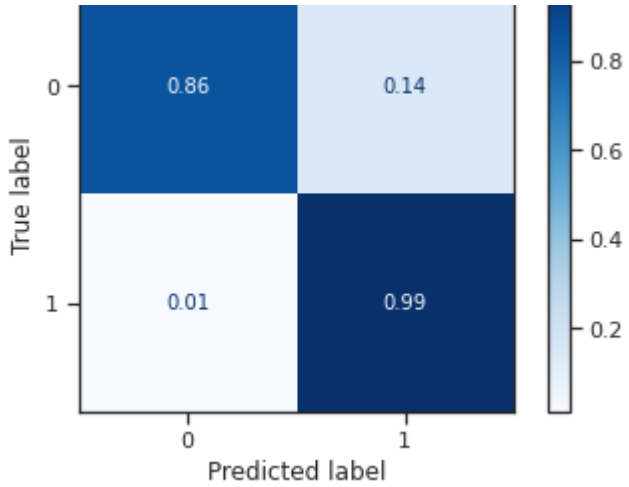
```
class MetricLogger:

    def __init__(self):
        self.df = pd.DataFrame(
            {'metric': pd.Series([], dtype='str'),
            'alg': pd.Series([], dtype='str'),
            'value': pd.Series([], dtype='float')})

    def add(self, metric, alg, value):
        """
        Добавление значения
        """
        # Удаление значения если оно уже было ранее добавлено
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].ind
        # Добавление нового значения
        temp = [{'metric':metric, 'alg':alg, 'value':value}]
        self.df = self.df.append(temp, ignore_index=True)

    def get_data_for_metric(self, metric, ascending=True):
        """
        Формирование данных с фильтром по метрике
        """
        temp_data = self.df[self.df['metric']==metric]
        temp_data_2 = temp_data.sort_values(by='value', ascending=ascending)
        return temp_data_2['alg'].values, temp_data_2['value'].values

    def plot(self, str_header, metric, ascending=True, figsize=(5, 5)):
        """
        Вывод графика
        """
        array_labels, array_metric = self.get_data_for_metric(metric, ascending)
        fig, ax1 = plt.subplots(figsize=figsize)
        pos = np.arange(len(array_metric))
```

```
        rects = ax1.barh(pos, array_metric,
                         align='center',
                         height=0.5,
                         tick_label=array_labels)
        ax1.set_title(str_header)
        for a,b in zip(pos, array_metric):
            plt.text(0.5, a-0.05, str(round(b,3)), color='white')
        plt.show()


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]))


labels=[1,2]
```

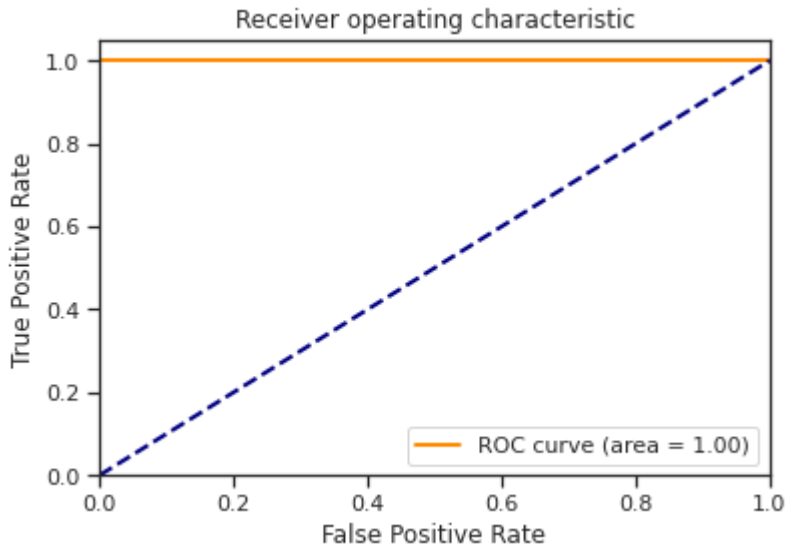
модели классификаторов

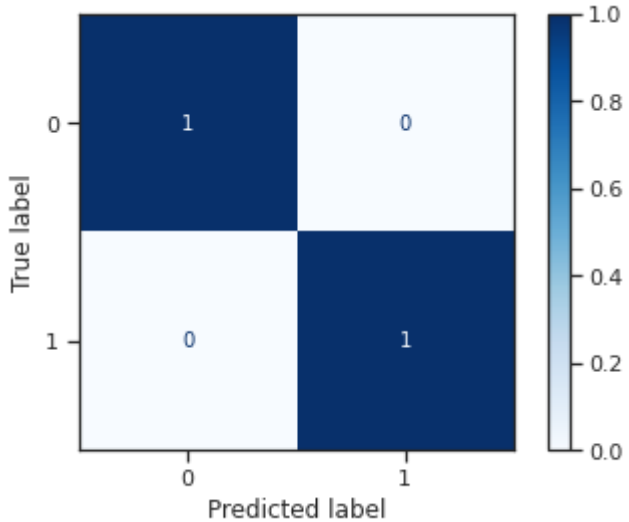
```
from sklearn import svm
clas_models = { 'LR': LogisticRegression(),
                'SVC':svm.SVC(decision_function_shape='ovo'),
                'Tree':DecisionTreeClassifier()}


clasMetricLogger = MetricLogger()


def draw_roc_curve(y_true, y_score, pos_label=1, average='micro'):
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()


def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(X_train, y_train)
    Y_pred = model.predict(X_test)
    print_accuracy_score_for_classes(y_test, Y_pred)
    precision = precision_score(y_test.values, Y_pred)
    recall = recall_score(y_test.values, Y_pred)
    f1 = f1_score(y_test.values, Y_pred)
    roc_auc = roc_auc_score(y_test.values, Y_pred)

    clasMetricLogger.add('precision', model_name, precision)
    clasMetricLogger.add('recall', model_name, recall)
    clasMetricLogger.add('f1', model_name, f1)
    clasMetricLogger.add('roc_auc', model_name, roc_auc)

    print('****************************************************')
    print(model)
    print('****************************************************')

    draw_roc_curve(y_test.values, Y_pred)

    plot_confusion_matrix(model,X_test, y_test.values,
                      display_labels=['0','1'],
                      cmap=plt.cm.Blues, normalize='true')
    plt.show()
```

```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

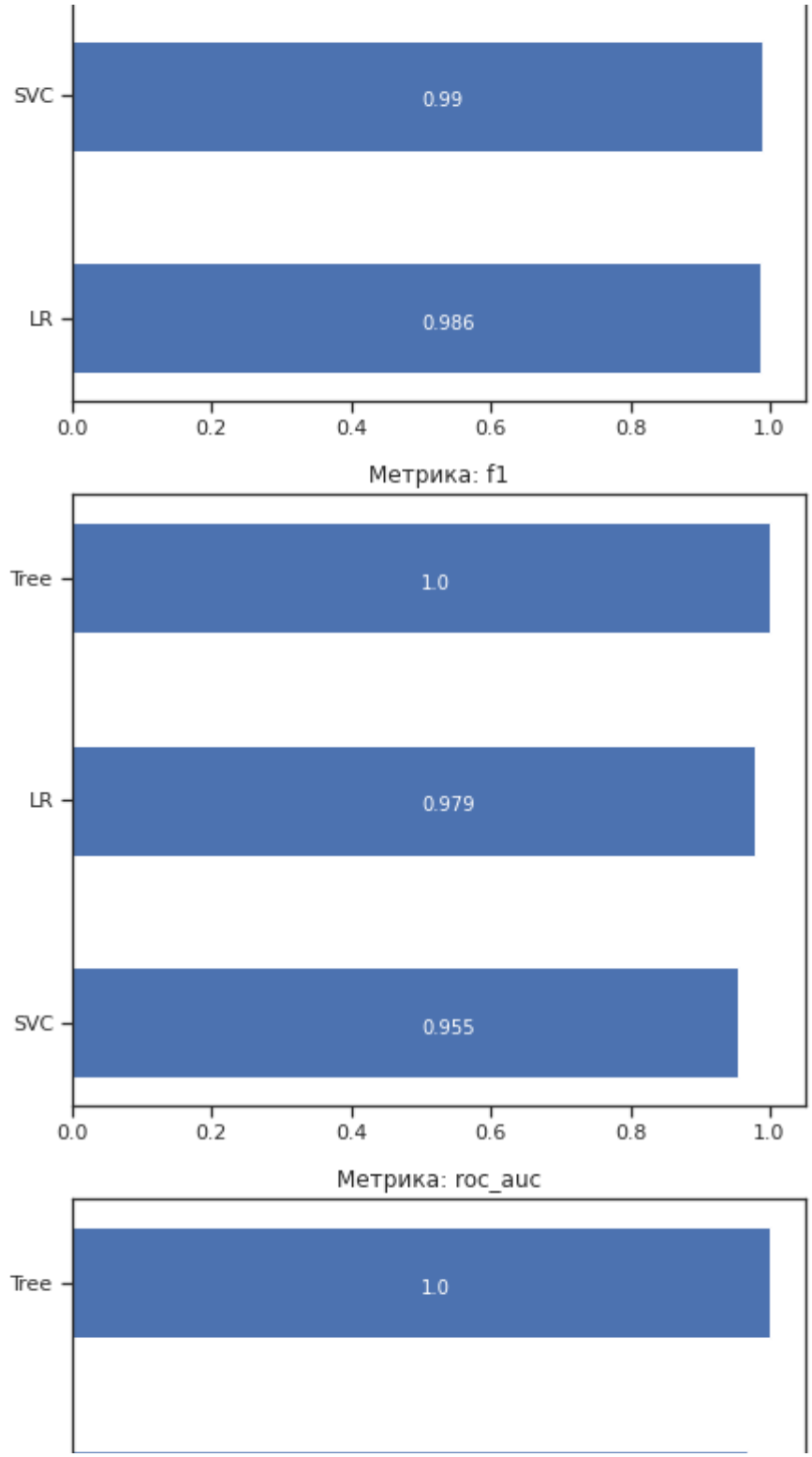



```
Метка      Accuracy
0.0        1.0
1.0        1.0
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
**************************************************
```





```
clas_metrics = clasMetricLogger.df['metric'].unique()


for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```



Grid Search для Logistic Regression:

```
clf = LogisticRegression()
grid_values = {'penalty': ['l1', 'l2'],'C':[0.001,.009,0.01,.09,1,5,10,25]}
grid_clf_acc = GridSearchCV(clf, param_grid = grid_values,scoring = 'recall')
```

```
grid_clf_acc.fit(X_train, y_train)
```

```
    GridSearchCV(cv=None, error_score=nan,
                 estimator=LogisticRegression(C=1.0, class_weight=None, dual=False
                                              fit_intercept=True,
                                              intercept_scaling=1, l1_ratio=None,
                                              max_iter=100, multi_class='auto',
                                              n_jobs=None, penalty='l2',
                                              random_state=None, solver='lbfgs',
                                              tol=0.0001, verbose=0,
                                              warm_start=False),
                 iid='deprecated', n_jobs=None,
                 param_grid={'C': [0.001, 0.009, 0.01, 0.09, 1, 5, 10, 25],
```

```
grid_clf_acc.best_estimator_
```

```
LogisticRegression(C=5, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

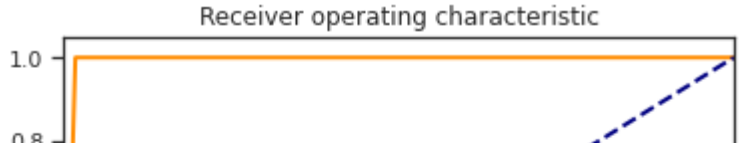
```
grid_clf_acc.best_params_
```

```
{'C': 5, 'penalty': 'l2'}
```

```
clas_models_grid = {'LR_5_12':grid_clf_acc.best_estimator_}
```

```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

```
Метка      Accuracy
0.0        0.9819277108433735
1.0        1.0
**************************************************
LogisticRegression(C=5, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
**************************************************
```



Grid Search для SVC:

```
param_grid = {'gamma':[1,0.1], 'kernel':['linear','rbf']}
```

```
grid = GridSearchCV(SVC(),param_grid=param_grid,refit = True, verbose=2)
grid.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV] gamma=1, kernel=linear ..........................................
[CV] ........................... gamma=1, kernel=linear, total=   0.0s
[CV] gamma=1, kernel=linear ..........................................
[CV] ........................... gamma=1, kernel=linear, total=   0.0s
[CV] gamma=1, kernel=linear ..........................................
[CV] ........................... gamma=1, kernel=linear, total=   0.0s
[CV] gamma=1, kernel=linear ..........................................
[CV] ........................... gamma=1, kernel=linear, total=   0.1s
[CV] gamma=1, kernel=linear ..........................................
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[CV] ........................... gamma=1, kernel=linear, total=   0.1s
[CV] gamma=1, kernel=rbf .............................................
[CV] ............................... gamma=1, kernel=rbf, total=   0.0s
[CV] gamma=1, kernel=rbf .............................................
[CV] ............................... gamma=1, kernel=rbf, total=   0.0s
[CV] gamma=1, kernel=rbf .............................................
[CV] ............................... gamma=1, kernel=rbf, total=   0.0s
[CV] gamma=1, kernel=rbf .............................................
```

```
grid.best_estimator_
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

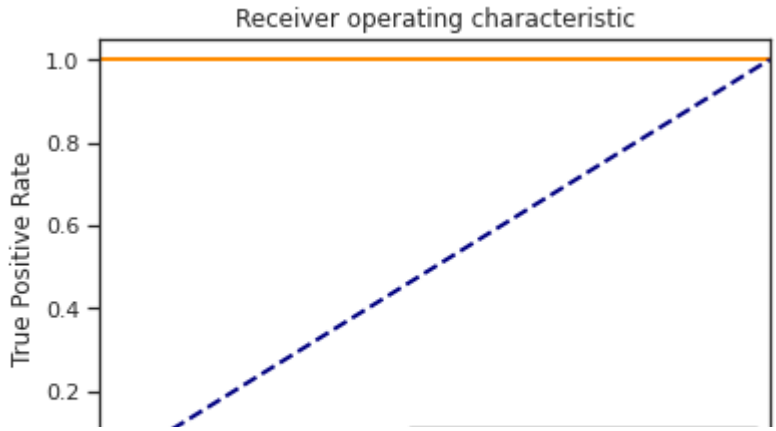
```
grid.best_params_
```

```
{'gamma': 1, 'kernel': 'linear'}
```

```
clas_models_grid = {'SVC_1':grid.best_estimator_}
```

```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

```
Метка      Accuracy
0.0        1.0
1.0        1.0
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
**************************************************
```



Grid Search для Decision Tree:

```
params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4]}
grid_search_cv = GridSearchCV(DecisionTreeClassifier(random_state=42), params, verb
grid_search_cv.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 294 candidates, totalling 882 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers
[Parallel(n_jobs=1)]: Done 882 out of 882 | elapsed:    2.9s finished
GridSearchCV(cv=3, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None
                                              criterion='gini', max_depth=None
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=42,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_leaf_nodes': [2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12
                                            13, 14, 15, 16, 17, 18, 19, 20, 21
                                            22, 23, 24, 25, 26, 27, 28, 29, 30
                                            31, ...],
                         'min_samples_split': [2, 3, 4]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

```
grid_search_cv.best_estimator_
```

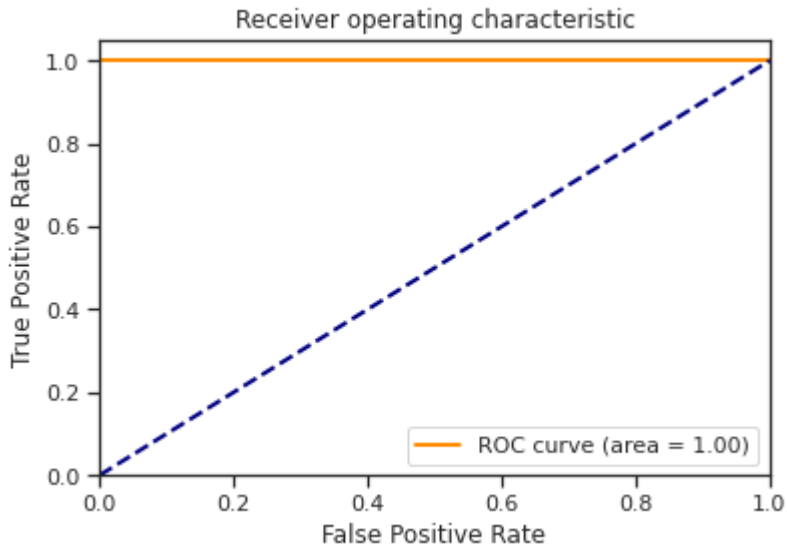
```
    DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                           max_depth=None, max_features=None, max_leaf_nodes=2,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, presort='deprecated',
```

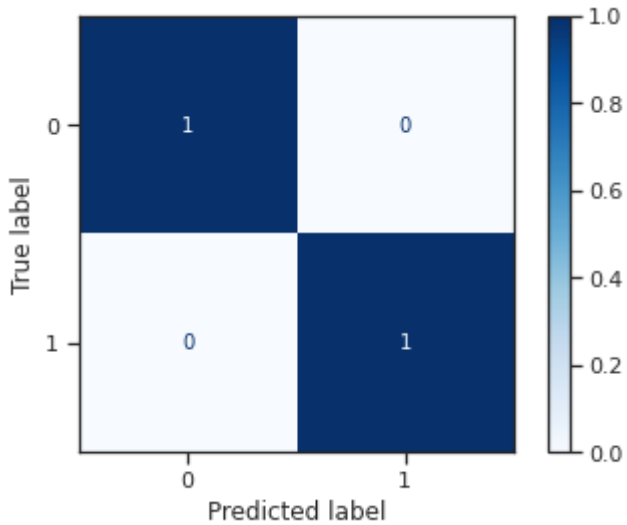
```
grid_search_cv.best_params_
```

```
{'max_leaf_nodes': 2, 'min_samples_split': 2}
```

```
clas_models_grid = {'Tree':grid_search_cv.best_estimator_}


for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

```
Метка      Accuracy
0.0        1.0
1.0        1.0
****************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=2,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
****************************************************
```





```
tree_cl=grid_search_cv.best_estimator_


from sklearn.tree.export import export_text
tree_rules = export_text(tree_cl, feature_names=list(data[['mean compactness','radi
                                                           'worst smoothness','wors
tree_rules
```

```
'|--- target <= 0.50\n|   |--- class: 0.0\n|--- target >  0.50\n|   |--- class
```

```
data_temp=data[['mean compactness','radius error','concave points error','worst tex
                                                           'worst smoothness','wors


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],[0,4],[0,5],[0,6],[0,7],
                                    [1,2],[1, 3],[1,4],[1,5],[1,6],[1,7],
                                    [2, 3],[2,4],[2,5],[2,6],[2,7],
                                    [3,4],[3,5],[3,6],[3,7],
                                    [4,5],[4,6],[4,7],
                                    [5,6],[5,7],[6,7]]):
        # We only take the two corresponding features
        temp=np.array(ds)
        X = temp[:, 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.columns[pair[0]])
        plt.ylabel(ds.columns[pair[1]])

        # Plot the training points
        for i, color in zip(range(n_classes), plot_colors):
            idx = np.where(y == i)
```