Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»

Кафедра ИУ5. Курс «Технологии машинного обучения»

Отчет по лабораторной работе №5: «Линейные модели, SVM и деревья решений»

Выполнил:
студент группы ИУ5-62
Андреев Артем

Подпись и дата:

Проверил:

Подпись и дата:

Москва, 2019 г.

In [2]:
```
import numpy as np
import pandas as pd
pd.set_option('display.max.rows', 1000)
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style='ticks')
```

Подготовка датасета

In [3]:
```
# House Sales in King County, USA
# Predict house price using regression
data = pd.read_csv('data/kc_house_data.csv')
data.shape
```

Out[3]:
```
(21613, 21)
```

In [4]:
```
data.head()
```

Out[4]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqft_above	sqft_basem
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	1180	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	2170	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	770	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	1050	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	1680	

5 rows × 21 columns

In [5]:
```
data = data.drop(columns=['id', 'date', ])
```

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

In [7]:
```
data.dtypes
```

Out[7]:
```
price            float64
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

In [8]:
```
# пропусков нет, разделим на обучающую и тестовую выборку
from sklearn.model_selection import train_test_split
```

In [9]:
```
# перед этим разделим исходный датасет на 2: один содержит независимые параметры, другой — зависимый (price)
X, y = data[data.columns[range(1, 19)]], data[data.columns[[0]]]
```

In [10]:
```
X.dtypes
```

Out[10]:
```
bedrooms           int64
bathrooms        float64
sqft_living        int64
sqft_lot           int64
floors           float64
waterfront         int64
view               int64
condition          int64
grade              int64
sqft_above         int64
sqft_basement      int64
yr_built           int64
yr_renovated       int64
zipcode            int64
lat              float64
long             float64
sqft_living15      int64
sqft_lot15         int64
dtype: object
```

In [11]:
```
y.dtypes
```

Out[11]:
```
price    float64
dtype: object
```

In [12]:
```
test_size = 0.2
state = 42
xTrain, xTest, yTrain, yTest = train_test_split(X, y, test_size=test_size, random_state=state)
len(xTrain), len(xTest), len(yTrain), len(yTest)
```

Out[12]:
```
(17290, 4323, 17290, 4323)
```

Обучение моделей

1) linear regression

In [13]:
```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(xTrain, yTrain)
```

Out[13]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

In [14]:
```
lin_reg.intercept_, lin_reg.coef_
```

Out[14]:
```
(array([6643873.52788867]),
 array([[-3.43354187e+04,  4.45645289e+04,  1.09015817e+02,
          8.88473539e-02,  7.00312952e+03,  5.62413070e+05,
          5.36411070e+04,  2.45267101e+04,  9.45678917e+04,
          7.00227408e+01,  3.89930757e+01, -2.68076890e+03,
          2.04156328e+01, -5.52253038e+02,  5.95968122e+05,
         -1.94585724e+05,  2.12143306e+01, -3.25831873e-01]]))
```

In [15]:
```
yPredictedTest = lin_reg.predict(xTest)
yPredictedTest
```

Out[15]:
```
array([[ 461209.94695865],
       [ 752443.51006947],
       [1238489.80205799],
       ...,
       [ 423101.46384868],
       [ 617785.6141686 ],
       [ 442344.46084995]])
```

In [16]:
```
yPredictedTrain = lin_reg.predict(xTrain)
yPredictedTrain
```

Out[16]:
```
array([[487301.30123506],
       [244429.83104153],
       [146433.18955254],
       ...,
       [456528.57446002],
       [-90768.72572574],
       [466824.07074714]])
```

In [17]:
```
yTest['price'].values
```

Out[17]:
```
array([ 365000.,  865000., 1038000., ...,  285000.,  605000.,  356500.])
```

In [18]:
```
# оценим качество модели регрессии
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# 1) mean absolute error – средняя абсолютная ошибка
print('mean_absolute_error (train): {}'.format(mean_absolute_error(yTrain, yPredictedTrain)))
print('mean_absolute_error (test): {}'.format(mean_absolute_error(yTest, yPredictedTest)))
```

```
mean_absolute_error (train): 125033.1649380497
mean_absolute_error (test): 127493.34208657558
```

In [19]:
```
# 2) mean squared error – средняя абсолютная ошибка
print('mean_squared_error (train): {}'.format(mean_squared_error(yTrain, yPredictedTrain)))
print('mean_squared_error (test): {}'.format(mean_squared_error(yTest, yPredictedTest)))
```

```
mean_squared_error (train): 39311882352.23276
mean_squared_error (test): 45173046132.78762
```

In [20]:
```
# 3) r^2 score
print('r^2 score (train): {}'.format(r2_score(yTrain, yPredictedTrain)))
print('r^2 score (test): {}'.format(r2_score(yTest, yPredictedTest)))
```

```
r^2 score (train): 0.6991021854487471
r^2 score (test): 0.7011904448878581
```

In [21]:
```
# from sklearn.linear_model import ElasticNet
```

2) SVM

In [22]:
```
# возьмем другой датасет, подходящий для задачи классификации
data_class = pd.read_csv('data/winequality-red.csv')
data_class.head()
```

Out[22]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

In [23]:
```
XC, yC = data_class[data_class.columns[range(11)]], data_class[data_class.columns[[11]]]
print('X:\n', XC.columns)
print('y:\n', yC.columns)
```

```
X:
 Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol'],
      dtype='object')
y:
 Index(['quality'], dtype='object')
```

In [24]:
```
from sklearn.preprocessing import Normalizer
normalizerX = Normalizer().fit(XC)
XC_n = normalizerX.transform(XC)
XC_n
```

Out[24]:
```
array([[0.19515252, 0.01846037, 0.        , ..., 0.09256559, 0.0147683 ,
        0.24789644],
       [0.10724124, 0.01209901, 0.        , ..., 0.04399641, 0.00934924,
        0.134739  ],
       [0.13545665, 0.01319834, 0.00069465, ..., 0.05661393, 0.01128805,
        0.17018912],
       ...,
       [0.12306863, 0.0099627 , 0.00253951, ..., 0.06680869, 0.01465103,
        0.21488174],
       [0.10566885, 0.01155193, 0.0021492 , ..., 0.06393861, 0.01271608,
        0.18268174],
       [0.12589937, 0.0065048 , 0.00986212, ..., 0.07113314, 0.01384893,
        0.2308155 ]])
```

In [25]:
```
test_size = 0.2
state = 42
xTrainC, xTestC, yTrainC, yTestC = train_test_split(XC_n, yC, test_size=test_size, random_state=state)
len(xTrainC), len(xTestC), len(yTrainC), len(yTestC)
```

Out[25]:
```
(1279, 320, 1279, 320)
```

In [26]:
```
from sklearn.svm import LinearSVC
lsvc = LinearSVC(C=1.0, max_iter=1000, verbose=10)
lsvc
```

Out[26]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=10)
```

In [27]:
```
# обучим
lsvc.fit(xTrainC, yTrainC.values.ravel())
```

```
[LibLinear]
```

Out[27]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=10)
```

In [28]:
```
yPredictedCTrain = lsvc.predict(xTrainC)
```

In [29]:
```
yPredictedCTest = lsvc.predict(xTestC)
```

In [30]:
```
# оценим качество модели классификации
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, f1_score

print('Accuracy (train): {} %'.format(accuracy_score(yTrainC, yPredictedCTrain) * 100))
print('Accuracy (test): {} %'.format(accuracy_score(yTestC, yPredictedCTest) * 100))
```

```
Accuracy (train): 52.61923377638781 %
Accuracy (test): 48.4375 %
```

In [31]:
```
print('Матрица ошибок: столбцы — предсказанное значение, строки — истинное значение')
print('Train\n', confusion_matrix(yTrainC, yPredictedCTrain))
print('Test\n', confusion_matrix(yTestC, yPredictedCTest))
```

```
Матрица ошибок: столбцы — предсказанное значение, строки — истинное значение
Train
 [[  0   0   5   4   0   0]
 [  0   0  19  23   1   0]
 [  0   0 380 168   3   0]
 [  0   0 215 289   2   0]
 [  0   0  30 123   4   0]
 [  0   0   2  10   1   0]]
Test
 [[ 0  0  0  1  0  0]
 [ 0  0  6  4  0  0]
 [ 0  0 83 46  1  0]
 [ 0  0 59 72  1  0]
 [ 0  0 10 32  0  0]
 [ 0  0  1  4  0  0]]
```

In [32]:
```
print('Train\n', precision_score(yTrainC, yPredictedCTrain, average='weighted'))
print('Test\n', precision_score(yTestC, yPredictedCTest, average='weighted'))
```

```
Train
 0.48141303911456856
Test
 0.39886006289308173

/Users/artyom.andreev/Study/.venv/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: Undefi
nedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [33]:
```
print('Train\n', f1_score(yTrainC, yPredictedCTrain, average='weighted'))
print('Test\n', f1_score(yTestC, yPredictedCTest, average='weighted'))
```

```
Train
 0.4818582987352344
Test
 0.4374714622052581

/Users/artyom.andreev/Study/.venv/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: Undefi
nedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

3) дерево

In [34]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [35]:
```
tree = DecisionTreeClassifier(random_state=state)
```

In [36]:
```
tree.fit(xTrainC, yTrainC)
```

Out[36]:
```
DecisionTreeClassifier(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=False, random_state=42,
            splitter='best')
```

In [37]:
```
list(zip(XC.columns.values, tree.feature_importances_))
```

Out[37]:
```
[('fixed acidity', 0.07592761108194683),
 ('volatile acidity', 0.14598227378069772),
 ('citric acid', 0.06964897539303043),
 ('residual sugar', 0.08764568627349723),
 ('chlorides', 0.06890346497441459),
 ('free sulfur dioxide', 0.06935533573614072),
 ('total sulfur dioxide', 0.10028831360946247),
 ('density', 0.07175757883546015),
 ('pH', 0.06242167374974539),
 ('sulphates', 0.10276118494928356),
 ('alcohol', 0.14530790161632096)]
```

In [38]:
```
yTreePredictedTrain = tree.predict(xTrainC)
```

In [39]:
```
yTreePredictedTest = tree.predict(xTestC)
```

In [40]:
```
# оценим качество модели классификации
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, f1_score

print('Accuracy (train): {} %'.format(accuracy_score(yTrainC, yTreePredictedTrain) * 100))
print('Accuracy (test): {} %'.format(accuracy_score(yTestC, yTreePredictedTest) * 100))
```

```
Accuracy (train): 100.0 %
Accuracy (test): 60.9375 %
```

In [41]:
```
print('Матрица ошибок: столбцы — предсказанное значение, строки — истинное значение')
print('Train\n', confusion_matrix(yTrainC, yTreePredictedTrain))
print('Test\n', confusion_matrix(yTestC, yTreePredictedTest))
```

```
Матрица ошибок: столбцы — предсказанное значение, строки — истинное значение
Train
 [[  9   0   0   0   0   0]
 [  0  43   0   0   0   0]
 [  0   0 551   0   0   0]
 [  0   0   0 506   0   0]
 [  0   0   0   0 157   0]
 [  0   0   0   0   0  13]]
Test
 [[ 0  1  0  0  0  0]
 [ 0  1  5  4  0  0]
 [ 0  3 86 36  5  0]
 [ 0  0 33 87 12  0]
 [ 0  0  3 14 21  4]
 [ 0  0  0  2  3  0]]
```

In [42]:
```
print('Train\n', precision_score(yTrainC, yTreePredictedTrain, average='weighted'))
print('Test\n', precision_score(yTestC, yTreePredictedTest, average='weighted'))
```

```
Train
 1.0
Test
 0.5995355734144864

/Users/artyom.andreev/Study/.venv/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: Undefi
nedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

In [43]:
```
print('Train\n', f1_score(yTrainC, yTreePredictedTrain, average='weighted'))
print('Test\n', f1_score(yTestC, yTreePredictedTest, average='weighted'))
```

```
Train
 1.0
Test
 0.6034694888503429

/Users/artyom.andreev/Study/.venv/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: Undefi
nedMetricWarning: F-score is ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

подбор одного гиперпараметра с использованием GridSearchCV и кросс-валидации.

2) SVM

In [84]:
```
from sklearn.model_selection import GridSearchCV, KFold, ShuffleSplit
parameters = [{'C': np.array(np.arange(0.1, 5.1, 0.1)),
               'max_iter': np.array([1000, 5000, 10000, 25000, 50000, 100000, 250000, 500000, 1000000])}]
lsvc_grid = GridSearchCV(LinearSVC(), parameters, cv=ShuffleSplit(n_splits=10, test_size=0.2), scoring='accur
acy',
                         n_jobs=-1,
                         verbose=10
                        )
lsvc_grid.fit(XC_n, yC.values.ravel())
```

```
Fitting 10 folds for each of 450 candidates, totalling 4500 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Batch computation too fast (0.0151s.) Setting batch_size=26.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.0s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    0.1s
[Parallel(n_jobs=-1)]: Done 250 tasks      | elapsed:    1.9s
[Parallel(n_jobs=-1)]: Batch computation too slow (2.0100s.) Setting batch_size=13.
[Parallel(n_jobs=-1)]: Done 484 tasks      | elapsed:    2.9s
[Parallel(n_jobs=-1)]: Done 770 tasks      | elapsed:    4.2s
[Parallel(n_jobs=-1)]: Done 952 tasks      | elapsed:    4.9s
[Parallel(n_jobs=-1)]: Done 1121 tasks      | elapsed:    6.0s
[Parallel(n_jobs=-1)]: Done 1290 tasks      | elapsed:    6.8s
[Parallel(n_jobs=-1)]: Done 1485 tasks      | elapsed:    8.2s
[Parallel(n_jobs=-1)]: Done 1680 tasks      | elapsed:    9.6s
[Parallel(n_jobs=-1)]: Done 1901 tasks      | elapsed:   12.0s
[Parallel(n_jobs=-1)]: Batch computation too slow (2.0137s.) Setting batch_size=6.
[Parallel(n_jobs=-1)]: Done 2101 tasks      | elapsed:   13.9s
[Parallel(n_jobs=-1)]: Done 2222 tasks      | elapsed:   15.1s
[Parallel(n_jobs=-1)]: Done 2336 tasks      | elapsed:   16.4s
[Parallel(n_jobs=-1)]: Done 2462 tasks      | elapsed:   17.9s
[Parallel(n_jobs=-1)]: Done 2588 tasks      | elapsed:   19.4s
[Parallel(n_jobs=-1)]: Done 2726 tasks      | elapsed:   21.3s
[Parallel(n_jobs=-1)]: Done 2864 tasks      | elapsed:   23.5s
[Parallel(n_jobs=-1)]: Done 3014 tasks      | elapsed:   25.6s
[Parallel(n_jobs=-1)]: Done 3164 tasks      | elapsed:   28.3s
[Parallel(n_jobs=-1)]: Done 3326 tasks      | elapsed:   30.7s
[Parallel(n_jobs=-1)]: Done 3488 tasks      | elapsed:   33.7s
[Parallel(n_jobs=-1)]: Done 3662 tasks      | elapsed:   36.6s
[Parallel(n_jobs=-1)]: Done 3836 tasks      | elapsed:   39.9s
[Parallel(n_jobs=-1)]: Done 4022 tasks      | elapsed:   43.6s
[Parallel(n_jobs=-1)]: Done 4208 tasks      | elapsed:   47.5s
[Parallel(n_jobs=-1)]: Batch computation too slow (2.0032s.) Setting batch_size=3.
[Parallel(n_jobs=-1)]: Done 4352 tasks      | elapsed:   50.2s
[Parallel(n_jobs=-1)]: Done 4451 tasks      | elapsed:   52.3s
[Parallel(n_jobs=-1)]: Done 4500 out of 4500 | elapsed:   53.2s finished
```

Out[84]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=10, random_state=None, test_size=0.2, train_size=None),
       error_score='raise-deprecating',
       estimator=LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid=[{'C': array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2, 1.3,
       1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6,
       2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9,
       4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 5. ]), 'max_iter': array([   1000,    5000,   1000
0,   25000,   50000,  100000,  250000,
        500000, 1000000])}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=10)
```

In [85]:
```
lsvc_grid.best_estimator_
```

Out[85]:
```
LinearSVC(C=5.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

In [86]:
```
lsvc_grid.best_score_
```

Out[86]:
```
0.535
```

In [87]:
```
lsvc_grid.best_params_
```

Out[87]:
```
{'C': 5.0, 'max_iter': 1000}
```

3) дерево

In [91]:
```
parameters = [{'random_state': np.array([state]),
               'max_depth': np.array([None, 10, 50, 100, 500, 1000, 5000, 10000]),
               'min_samples_split': np.array(range(2, 11)),
               'min_samples_leaf': np.array(range(1, 11))
              }]
tree_grid = GridSearchCV(DecisionTreeClassifier(), parameters, cv=ShuffleSplit(n_splits=10, test_size=0.2), s
coring='accuracy',
                         n_jobs=-1,
                         verbose=10
                        )
tree_grid.fit(XC_n, yC.values.ravel())
```

```
Fitting 10 folds for each of 720 candidates, totalling 7200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    0.7s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Batch computation too fast (0.1732s.) Setting batch_size=2.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    0.8s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    0.9s
[Parallel(n_jobs=-1)]: Batch computation too fast (0.1545s.) Setting batch_size=4.
[Parallel(n_jobs=-1)]: Done  52 tasks      | elapsed:    1.0s
[Parallel(n_jobs=-1)]: Batch computation too fast (0.1130s.) Setting batch_size=14.
[Parallel(n_jobs=-1)]: Done  78 tasks      | elapsed:    1.1s
[Parallel(n_jobs=-1)]: Done 130 tasks      | elapsed:    1.2s
[Parallel(n_jobs=-1)]: Done 302 tasks      | elapsed:    1.7s
[Parallel(n_jobs=-1)]: Done 512 tasks      | elapsed:    2.1s
[Parallel(n_jobs=-1)]: Done 722 tasks      | elapsed:    2.6s
[Parallel(n_jobs=-1)]: Done 960 tasks      | elapsed:    3.0s
[Parallel(n_jobs=-1)]: Done 1198 tasks      | elapsed:    3.4s
[Parallel(n_jobs=-1)]: Done 1464 tasks      | elapsed:    3.9s
[Parallel(n_jobs=-1)]: Done 1730 tasks      | elapsed:    4.4s
[Parallel(n_jobs=-1)]: Done 2024 tasks      | elapsed:    4.9s
[Parallel(n_jobs=-1)]: Done 2318 tasks      | elapsed:    5.6s
[Parallel(n_jobs=-1)]: Done 2640 tasks      | elapsed:    6.1s
[Parallel(n_jobs=-1)]: Done 2962 tasks      | elapsed:    6.8s
[Parallel(n_jobs=-1)]: Done 3312 tasks      | elapsed:    7.5s
[Parallel(n_jobs=-1)]: Done 3662 tasks      | elapsed:    8.1s
[Parallel(n_jobs=-1)]: Done 4040 tasks      | elapsed:    8.9s
[Parallel(n_jobs=-1)]: Done 4418 tasks      | elapsed:    9.6s
[Parallel(n_jobs=-1)]: Done 4824 tasks      | elapsed:   10.4s
[Parallel(n_jobs=-1)]: Done 5230 tasks      | elapsed:   11.4s
[Parallel(n_jobs=-1)]: Done 5664 tasks      | elapsed:   12.3s
[Parallel(n_jobs=-1)]: Done 6098 tasks      | elapsed:   13.1s
[Parallel(n_jobs=-1)]: Done 6560 tasks      | elapsed:   14.1s
[Parallel(n_jobs=-1)]: Done 7200 out of 7200 | elapsed:   15.3s finished
```

Out[91]:
```
GridSearchCV(cv=ShuffleSplit(n_splits=10, random_state=None, test_size=0.2, train_size=None),
       error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(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=False, random_state=None,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=-1,
       param_grid=[{'random_state': array([42]), 'max_depth': array([None, 10, 50, 100, 500, 1000, 5000, 100
00], dtype=object), 'min_samples_split': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10]), 'min_samples_leaf': ar
ray([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='accuracy', verbose=10)
```

In [92]:
```
tree_grid.best_estimator_
```

Out[92]:
```
DecisionTreeClassifier(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=False, random_state=42,
            splitter='best')
```

In [94]:
```
tree_grid.best_score_
```

Out[94]:
```
0.5975
```

In [95]:
```
tree_grid.best_params_
```

Out[95]:
```
{'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'random_state': 42}
```

Обучим снова с найденными оптимальными гиперпараметрами

In [88]:
```
lsvc_grid.best_estimator_.fit(xTrainC, yTrainC.values.ravel())

yPredictedCTrainNew = lsvc_grid.best_estimator_.predict(xTrainC)
yPredictedCTestNew = lsvc_grid.best_estimator_.predict(xTestC)

print('Accuracy (train): {} %'.format(accuracy_score(yTrainC, yPredictedCTrain) * 100))
print('Accuracy (test): {} %'.format(accuracy_score(yTestC, yPredictedCTest) * 100))

print('New accuracy (train): {} %'.format(accuracy_score(yTrainC, yPredictedCTrainNew) * 100))
print('New accuracy (test): {} %'.format(accuracy_score(yTestC, yPredictedCTestNew) * 100))
```

```
Accuracy (train): 52.61923377638781 %
Accuracy (test): 48.4375 %
New accuracy (train): 55.355746677091474 %
New accuracy (test): 50.625 %
```

In [97]:
```
tree_grid.best_estimator_.fit(xTrainC, yTrainC.values.ravel())

yTreePredictedTrainNew = tree_grid.best_estimator_.predict(xTrainC)
yTreePredictedTestNew = tree_grid.best_estimator_.predict(xTestC)

print('Accuracy (train): {} %'.format(accuracy_score(yTrainC, yTreePredictedTrain) * 100))
print('Accuracy (test): {} %'.format(accuracy_score(yTestC, yTreePredictedTest) * 100))

print('New accuracy (train): {} %'.format(accuracy_score(yTrainC, yTreePredictedTrainNew) * 100))
print('New accuracy (test): {} %'.format(accuracy_score(yTestC, yTreePredictedTestNew) * 100))
```

```
Accuracy (train): 100.0 %
Accuracy (test): 60.9375 %
New accuracy (train): 100.0 %
New accuracy (test): 60.9375 %
```

так как дефолтные параметры и оказались оптимальными

In [ ]:
```
```