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 ИУ5 «Системы обработки информации и управления»

Курсовая работа

по дисциплине «Технологии машинного обучения»

на тему:

Исследование датасета. Решение задачи классификации

Выполнили:
студент группы № ИУ5-62
Миронов С.В.
подпись, дата

Проверил:
Гапанюк Ю.Е.
подпись, дата

2019 г.

Задание

1. Поиск и выбор набора данных для построения моделей машинного обучения. Построение модели машинного обучения и решение задачи классификации.
2. Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.
3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.
4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.
5. Выбор метрик для последующей оценки качества моделей.
6. Выбор наиболее подходящих моделей для решения задачи классификации.
7. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров. Производится обучение моделей на основе обучающей выборки и оценка качества моделей на основе тестовой выборки.
8. Подбор гиперпараметров для выбранных моделей.
9. Формирование выводов о качестве построенных моделей на основе выбранных метрик

Оглавление

Введение

Данная работа предназначена для усвоения и закрепления знаний по дисциплине «Технологии машинного обучения». Здесь закрепляются навыки проведения разведочного анализа данных, выбора признаков для построения модели, проведения корреляционного анализа, подбора метрик, решения задачи регрессии, построения базового решения и подбора гиперпараметров.

~~Lab1_end~~

June 3, 2019

```
In [1]: !pip install seaborn
```

```
DEPRECATION: Python 2.7 will reach the end of its life on January 1st, 2020. Please upgrade you
Requirement already satisfied: seaborn in /usr/local/lib/python2.7/dist-packages (0.9.0)
Requirement already satisfied: matplotlib>=1.4.3 in /usr/lib/python2.7/dist-packages (from seal
Requirement already satisfied: numpy>=1.9.3 in /usr/local/lib/python2.7/dist-packages (from sea
Requirement already satisfied: pandas>=0.15.2 in /usr/local/lib/python2.7/dist-packages (from s
Requirement already satisfied: scipy>=0.14.0 in /usr/local/lib/python2.7/dist-packages (from se
Requirement already satisfied: pytz>=2011k in /usr/lib/python2.7/dist-packages (from pandas>=0
Requirement already satisfied: python-dateutil>=2.5.0 in /usr/lib/python2.7/dist-packages (fro
```

```
In [2]: import numpy as np
        import pandas as pd
        from sklearn import datasets
        import matplotlib.pyplot as plt
        import seaborn as sns

        %matplotlib inline
```

```
In [3]: df = pd.read_csv('heart.csv', sep=",")
```

1 Data Set Information:

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

1.1 Attribute Information:

1. age
2. sex
3. chest pain type (4 values)
4. resting blood pressure
5. serum cholestoral in mg/dl
6. fasting blood sugar > 120 mg/dl
7. resting electrocardiographic results (values 0,1,2)

8. maximum heart rate achieved
9. exercise induced angina
10. oldpeak = ST depression induced by exercise relative to rest
11. the slope of the peak exercise ST segment
12. number of major vessels (0-3) colored by flourosopy
13. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect

```
In [4]: df.shape

Out[4]: (303, 14)

In [5]: df.head()

Out[5]:    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  slope  \
        0   63    1   3       145   233    1        0      150      0      2.3      0
        1   37    1   2       130   250    0        1      187      0      3.5      0
        2   41    0   1       130   204    0        0      172      0      1.4      2
        3   56    1   1       120   236    0        1      178      0      0.8      2
        4   57    0   0       120   354    0        1      163      1      0.6      2

           ca  thal  target
        0   0     1       1
        1   0     2       1
        2   0     2       1
        3   0     2       1
        4   0     2       1

In [6]: df.dtypes

Out[6]: age           int64
        sex           int64
        cp            int64
        trestbps      int64
        chol          int64
        fbs           int64
        restecg       int64
        thalach       int64
        exang         int64
        oldpeak     float64
        slope         int64
        ca            int64
        thal          int64
        target        int64
        dtype: object
```

, :

```
In [7]: df.isnull().sum()
```

```
Out[7]: age         0
        sex         0
        cp          0
        trestbps    0
        chol        0
        fbs         0
        restecg     0
        thalach     0
        exang       0
        oldpeak     0
        slope       0
        ca          0
        thal        0
        target      0
        dtype: int64

    .
    :

In [8]: df['target'].unique()

Out[8]: array([1, 0])


In [9]: df.describe()

Out[9]:                age         sex          cp    trestbps        chol         fbs  \
        count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
        mean    54.366337    0.683168    0.966997  131.623762  246.264026    0.148515
        std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356198
        min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000000
        25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000000
        50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000000
        75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000000
        max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000000

                  restecg     thalach       exang     oldpeak       slope          ca  \
        count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000000
        mean     0.528053  149.646865    0.326733    1.039604    1.399340    0.729373
        std      0.525860   22.905161    0.469794    1.161075    0.616226    1.022606
        min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000000
        25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000000
        50%      1.000000  153.000000    0.000000    0.800000    1.000000    0.000000
        75%      1.000000  166.000000    1.000000    1.600000    2.000000    1.000000
        max      2.000000  202.000000    1.000000    6.200000    2.000000    4.000000

                    thal      target
        count  303.000000  303.000000
```

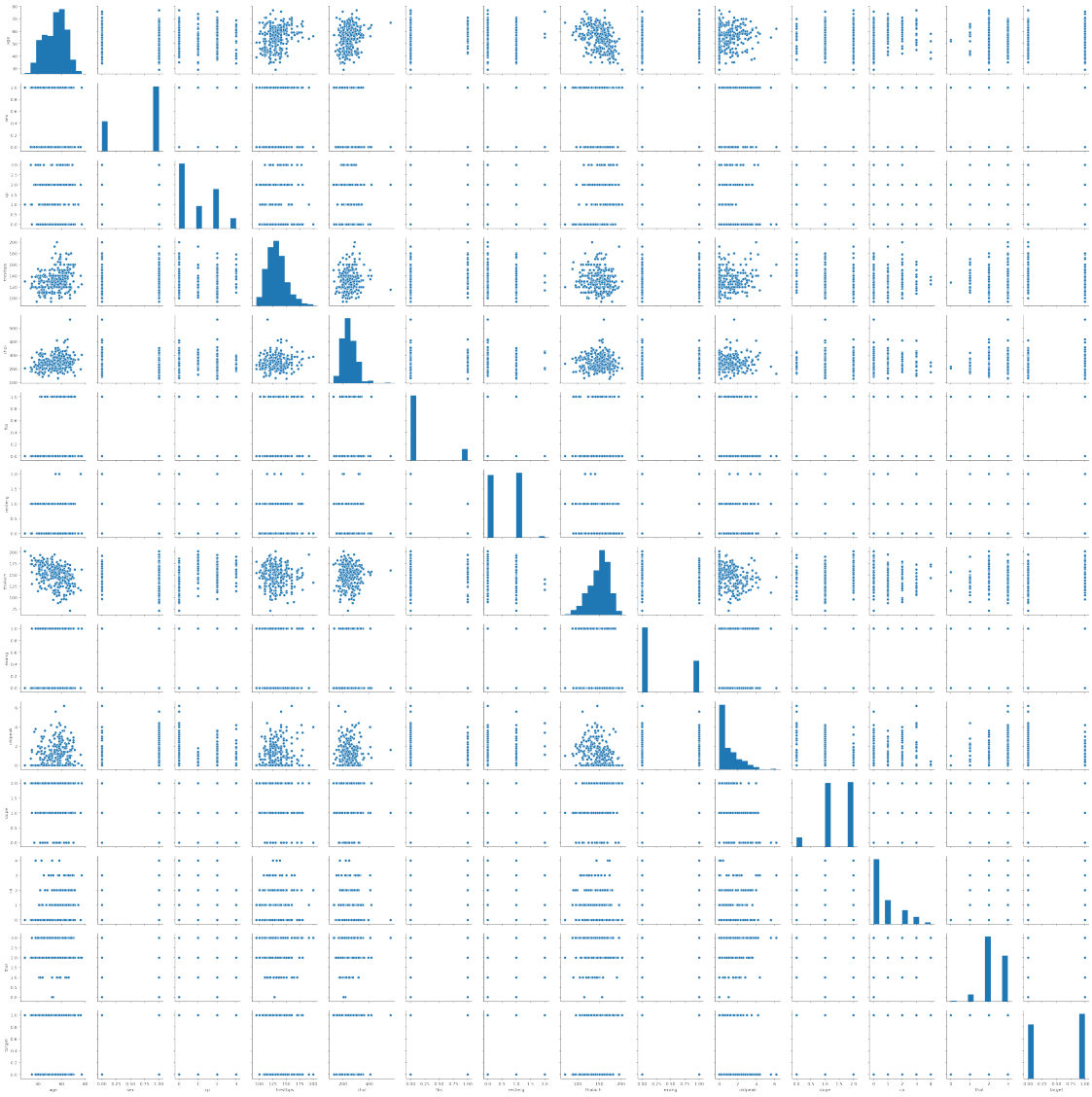
```
mean      2.313531    0.544554
std       0.612277    0.498835
min       0.000000    0.000000
25%       2.000000    0.000000
50%       2.000000    1.000000
75%       3.000000    1.000000
max       3.000000    1.000000
```

-,

In [10]: sns.pairplot(data= df)

Out[10]: <seaborn.axisgrid.PairGrid at 0x7f2287a89940>

Out[10]:



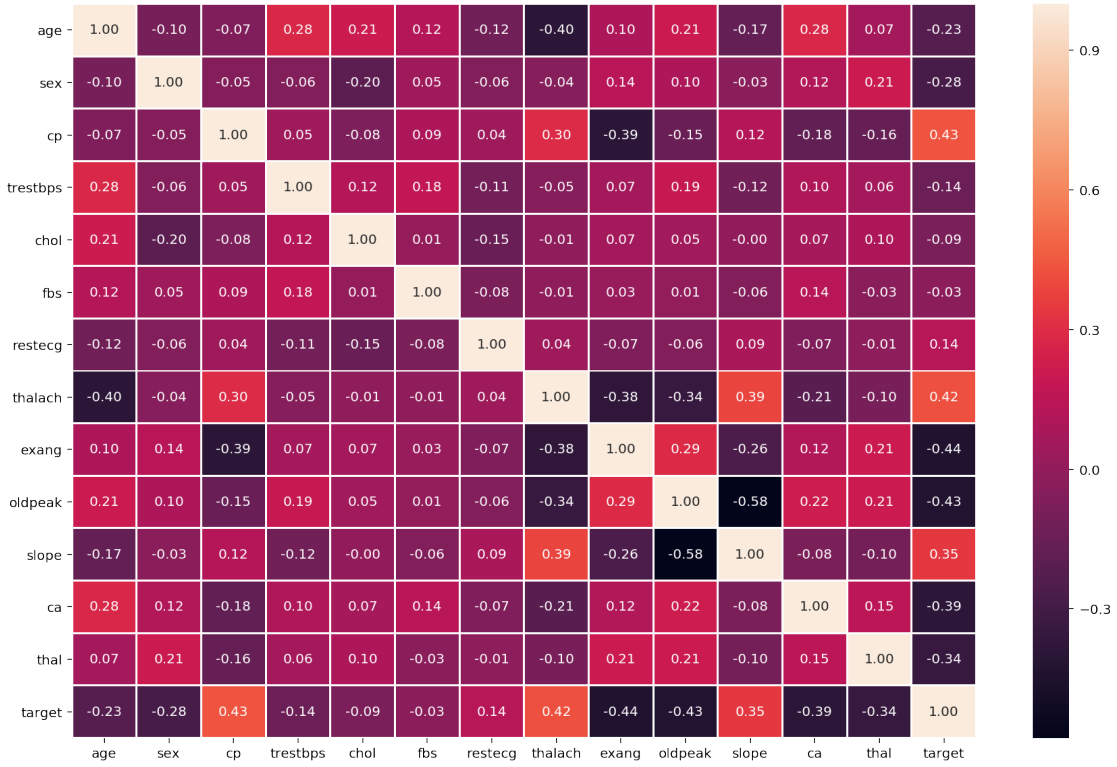
In [11]: *#sns.pairplot(df, hue= "target")*

:

In [12]:
```
plt.figure(figsize=(15, 10))
sns.heatmap(df.corr(),annot=True, fmt='.2f', linewidths=1)
```

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x7f228082e0f0>

Out[12]:



In [13]:
```
plt.figure(figsize=(15, 10))
sns.heatmap(df.corr(method='spearman'),annot=True, fmt='.3f')
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x7f227ed99390>

Out[13]:


age 1.000 -0.099 -0.087 0.286 0.196 0.114 -0.133 -0.398 0.090 0.268 -0.184 0.341 0.087 -0.238
sex -0.099 1.000 -0.062 -0.053 -0.151 0.045 -0.048 -0.040 0.142 0.101 -0.025 0.119 0.251 -0.281
cp -0.087 -0.062 1.000 0.035 -0.092 0.090 0.066 0.324 -0.418 -0.161 0.159 -0.216 -0.208 0.461
trestbps 0.286 -0.053 0.035 1.000 0.127 0.152 -0.126 -0.040 0.053 0.154 -0.087 0.090 0.060 -0.122
chol 0.196 -0.151 -0.092 0.127 1.000 0.018 -0.162 -0.047 0.092 0.045 -0.013 0.112 0.084 -0.121
fbs 0.114 0.045 0.090 0.152 0.018 1.000 -0.082 -0.014 0.026 0.028 -0.046 0.135 -0.007 -0.028
restecg -0.133 -0.048 0.066 -0.126 -0.162 -0.082 1.000 0.088 -0.077 -0.077 0.114 -0.098 -0.011 0.149
thalach -0.398 -0.040 0.324 -0.040 -0.047 -0.014 0.088 1.000 -0.401 -0.433 0.437 -0.257 -0.161 0.428
exang 0.090 0.142 -0.418 0.053 0.092 0.026 -0.077 -0.401 1.000 0.297 -0.274 0.162 0.247 -0.437
oldpeak 0.268 0.101 -0.161 0.154 0.045 0.028 -0.077 -0.433 0.297 1.000 -0.595 0.225 0.255 -0.421
slope -0.184 -0.025 0.159 -0.087 -0.013 -0.046 0.114 0.437 -0.274 -0.595 1.000 -0.100 -0.155 0.371
ca 0.341 0.119 -0.216 0.090 0.112 0.135 -0.098 -0.257 0.162 0.225 -0.100 1.000 0.189 -0.458
thal 0.087 0.251 -0.208 0.060 0.084 -0.007 -0.011 -0.161 0.247 0.255 -0.155 0.189 1.000 -0.403
target -0.238 -0.281 0.461 -0.122 -0.121 -0.028 0.149 0.428 -0.437 -0.421 0.371 -0.458 -0.403 1.000
age sex cp trestbps chol fbs restecg thalach exang oldpeak slope ca thal target
0.9
0.6
0.3
0.0
−0.3

Построение различных моделей:

1)Модель K ближайших соседей(K=13) – точность мала: 67,5%

In [21]:
```
random_search = GridSearchCV(estimator= KNeighborsClassifier(),
                             param_grid= {'n_neighbors': [5,10,12,13,14,15,16,20,30,50]},
                             scoring= 'f1_weighted',
                             cv= 3)

random_search.fit(df.loc[:, df.columns != 'target'], df['target'])
```

Out[21]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=None, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_neighbors': [5, 10, 12, 13, 14, 15, 16, 20, 30, 50]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='f1_weighted', verbose=0)
```

In [22]:
```
random_search.best_params_
```

Out[22]:
```
{'n_neighbors': 13}
```

In [23]:
```
random_search.best_score_
```

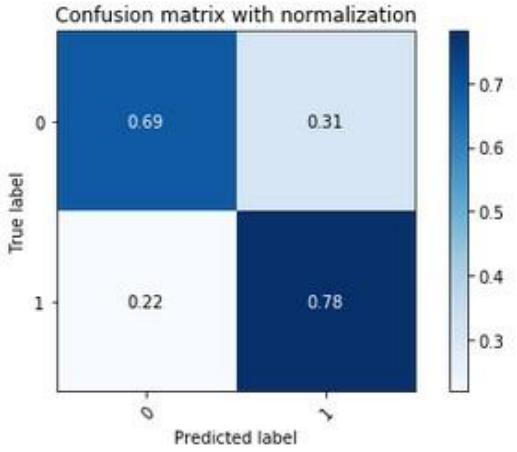
Out[23]:
```
0.6752523247180728
```

In [24]:
```
KNeighborsClassifierObj = KNeighborsClassifier(n_neighbors=13)
KNeighborsClassifierObj.fit(X_train, y_train)
result_y2=KNeighborsClassifierObj.predict(X_test)
```

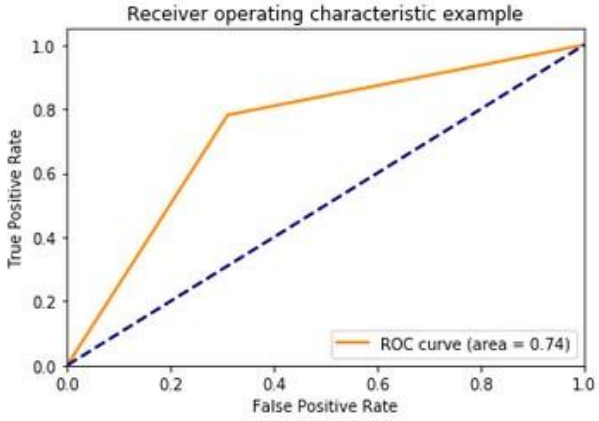
In [25]:
```
plot_confusion_matrix(y_test, result_y2,
                      classes=np.array(['0', '1']),
                      normalize=True,
                      title='Confusion matrix with normalization')
```

```
Normalized confusion matrix
<matplotlib.axes._subplots.AxesSubplot at 0x7f185581b5f8>
```

Out[25]:



In [26]:
```
draw_roc_curve(y_test, result_y2, pos_label=1, average='micro')
```



2) Линейная модель(кросс-валидация для итераций)- точность 83,6%

In [36]:
```
clas = SGDClassifier()
param = {'max_iter':range(1,5000,500)}
GV = GridSearchCV(clas, param, cv=3)
GV.fit(X_train, y_train)
```

Out[36]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=None,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'max_iter': range(1, 5000, 50)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [37]:
```
GV.best_estimator_
```

Out[37]:
```
SGDClassifier(alpha=0.0001, average=False, class_weight=None,
       early_stopping=False, epsilon=0.1, eta0=0.0, fit_intercept=True,
       l1_ratio=0.15, learning_rate='optimal', loss='hinge', max_iter=1801,
       n_iter=None, n_iter_no_change=5, n_jobs=None, penalty='l2',
       power_t=0.5, random_state=None, shuffle=True, tol=None,
       validation_fraction=0.1, verbose=0, warm_start=False)
```

In [38]:
```
print(accuracy_score(GV.predict(X_test), y_test))
```

```
0.8360655737704918
```

3) Модель опорных векторов(кросс-валидация для кол-ва итераций) – точность 88,5%

In [42]:
```
clas = LinearSVC()
param = {'max_iter':range(100,20000,1000)}
GV = GridSearchCV(clas, param, cv=3)
GV.fit(X_train, y_train)
```

Out[42]:
```
GridSearchCV(cv=3, 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=None,
       param_grid={'max_iter': range(100, 20000, 1000)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [43]:
```
GV.best_estimator_
```

Out[43]:
```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=16100,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0)
```

In [44]:
```
print(accuracy_score(GV.predict(X_test), y_test))
```

```
0.8852459016393442
```

4) Модель деревьев решений(кросс-валидация по глубине дерева) – точность 81,9%

In [30]:
```
clas = DecisionTreeClassifier()
param = {'max_depth':range(1,30)}
GV = GridSearchCV(clas, param, cv=3)
GV.fit(X_train, y_train)
```

Out[30]:
```
GridSearchCV(cv=3, 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=None,
       param_grid={'max_depth': range(1, 30)}, pre_dispatch='2*n_jobs',
       refit=True, return_train_score='warn', scoring=None, verbose=0)
```

In [31]:
```
GV.best_estimator_
```

Out[31]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            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')
```

In [32]:
```
print(accuracy_score(GV.predict(X_test), y_test))
```

```
0.819672131147541
```

5) Ансамблевая модель случайного леса с кросс-вал. – точность 85,2%

In [52]:
```
clas =  RandomForestClassifier()
param = {'n_estimators':range(10,300,10)}
GV = GridSearchCV(clas, param, cv=3)
GV.fit(X_train, y_train)
```

Out[52]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': range(10, 300, 10)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [53]:
```
GV.best_estimator_
```

Out[53]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', 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, n_estimators=160, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [54]:
```
print(accuracy_score(GV.predict(X_test), y_test))
```

```
0.8524590163934426
```

6) Ансамблевая модель градиентного бустинга – точность 83,6%

In [62]:
```
clas = GradientBoostingClassifier()
param = {'n_estimators':range(10,500,10)}
GV = GridSearchCV(clas, param, cv=3)
GV.fit(X_train, y_train)
```

Out[62]:
```
GridSearchCV(cv=3, error_score='raise-deprecating',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_sampl...      subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': range(10, 500, 10)},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [63]:
```
GV.best_estimator_
```

Out[63]:
```
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              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, n_estimators=50,
              n_iter_no_change=None, presort='auto', random_state=None,
              subsample=1.0, tol=0.0001, validation_fraction=0.1,
              verbose=0, warm_start=False)
```

In [64]:
```
print(accuracy_score(GV.predict(X_test), y_test))
```

```
0.8360655737704918
```

Заключение

В данной работе были закреплены навыки проведения разведочного анализа данных, выбора признаков для построения модели, проведения корреляционного анализа, подбора метрик, решения задачи регрессии, построения базового решения и подбора гиперпараметров, сравнены качества моделей. Из-за малой выборки точность предсказания получается очень нестабильной. Но лучшая точность в моих экспериментах достигается на модели опорных векторов.

Список использованной литературы

1. https://github.com/ugapanyuk/ml_course/wiki/COURSE_TMO
2. https://lightgbm.readthedocs.io/en/latest/
3. https://www.mlflow.org/
4. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html
5. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error.html