Московский государственный технический университет им. Н.Э. Баумана
Кафедра «Системы обработки информации и управления»

Домашняя работа
по дисциплине
«Методы машинного обучения»

Выполнил:
студент группы ИУ5-21М
Якубов А. Р.

Москва — 2019 г.

1. Цель задания

Решение комплексной задачи машинного обучения.

2. Задание

1. Поиск и выбор набора данных для построения моделей машинного обучения. На основе выбранного набора данных студент должен построить модели машинного обучения для решения или задачи классификации, или задачи регрессии.
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. В зависимости от набора данных, порядок выполнения пунктов 2, 3, 4 может быть изменен.
5. Выбор метрик для последующей оценки качества моделей. Необходимо выбрать не менее двух метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее трех моделей, хотя бы одна из которых должна быть ансамблевой.
7. Формирование обучающей и тестовой выборок на основе исходного набора данных.
8. 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.
9. Подбор гиперпараметров для выбранных моделей. Рекомендуется подбирать не более 1-2 гиперпараметров. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.
10. Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.
11. Формирование выводов о качестве построенных моделей на основе выбранных метрик.

3. Ход выполнения

3.1. Загрузка данных

В качестве исходных данных для решения поставленной задачи был выбран датасет Heart Disease UCI (https://www.kaggle.com/ronitf/heart-disease-uci). Данный набор содержит 303 записи, 14 признаков, целевой признак относится к наличию болезни сердца у пациента: 0 - нет болезни сердца, 1 - есть. На основе данного датасета будем производить построение модели для решения задачи классификации.

```
In [46]: import warnings
         warnings.filterwarnings('ignore')
```

```
In [47]: import os
         import numpy as np
         import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
#os.listdir()
data = pd.read_csv('heart.csv',
                   sep=",", encoding="iso-8859-1")
```

3.2. Разведочный анализ данных

```
In [48]: data.head()
```

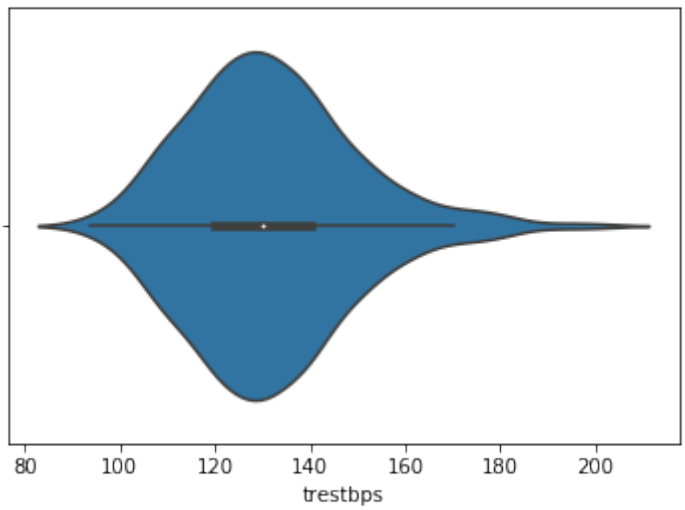
```
Out[48]:    age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpea
         0   63    1   3       145   233    1        0      150      0      2.
         1   37    1   2       130   250    0        1      187      0      3.
         2   41    0   1       130   204    0        0      172      0      1.
         3   56    1   1       120   236    0        1      178      0      0.
         4   57    0   0       120   354    0        1      163      1      0.

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

```
In [49]: data.shape
```

```
Out[49]: (303, 14)
```

```
In [50]: sns.violinplot(x=data['trestbps'])
```

```
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x1580ab2cf28>
```



```
In [51]: total_count = data.shape[0]
         num_cols = []
         for col in data.columns:
             # Количество пустых значений
             temp_null_count = data[data[col].isnull()].shape[0]
             dt = str(data[col].dtype)
             if temp_null_count>0:
                 num_cols.append(col)
                 temp_perc = round((temp_null_count / total_count) * 100.0, 2)
                 print('Колонка {}. Тип данных {}. Количество пустых значений
                       .format(col, dt, temp_null_count, temp_perc))

         data_cleared = data
```

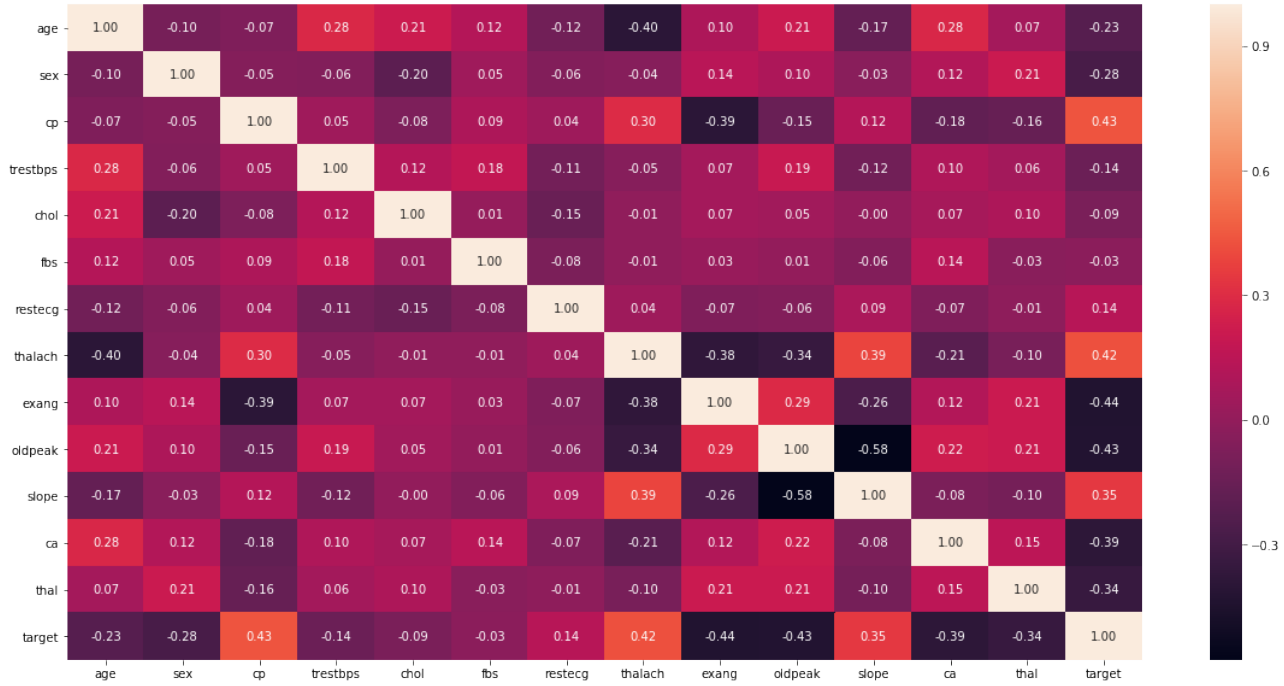
Пропусков в данных обнаружено не было.

```
In [52]: data.dtypes

Out[52]: 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
```

В модели отсутствуют категориальные признаки, поэтому нет необходимости проводить кодирование. В качетсве признака для классификации выберем предлагаемый признак target - наличие у пациента сердечных заболеваний.

3.3. Проведение корреляционного анализа данных

```
In [53]: fig, ax = plt.subplots(figsize=(20,10))
         sns.heatmap(data.corr(), annot=True, fmt='.2f', ax=ax)

Out[53]: <matplotlib.axes._subplots.AxesSubplot at 0x1580ace9518>
```

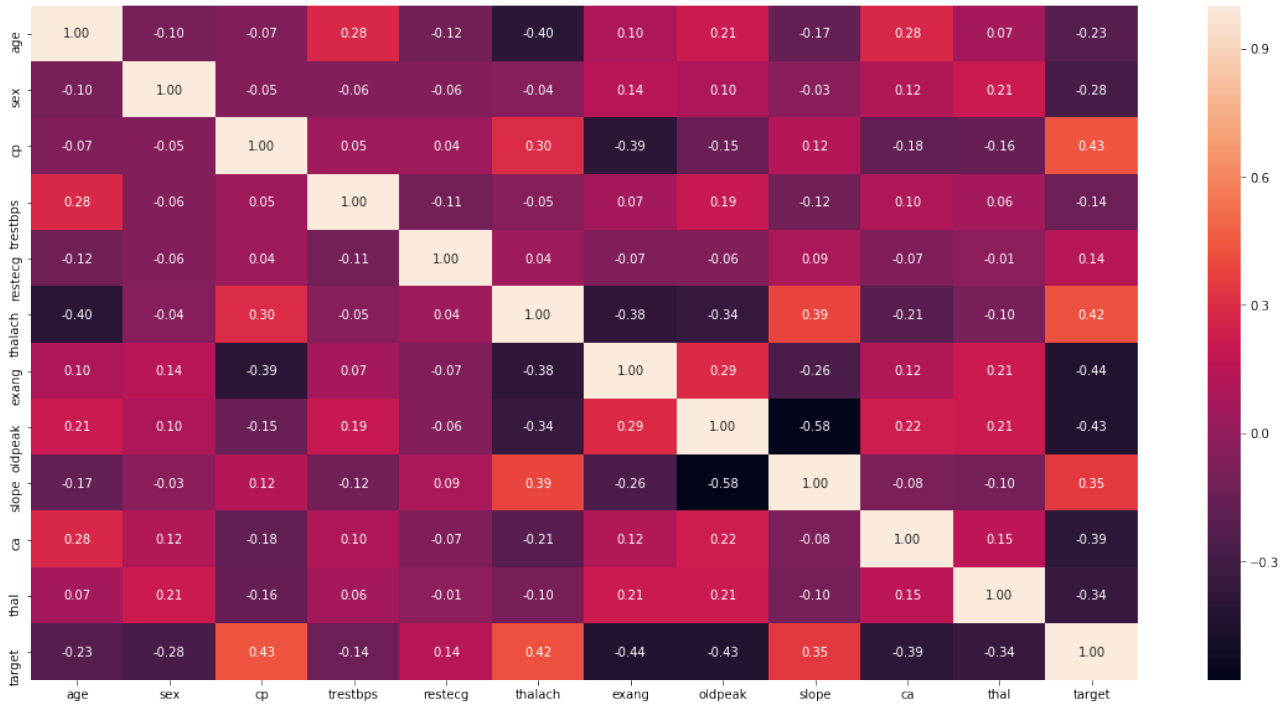


В результате построения корреляционной матрицы было выявлено, что признаки fbs (fasting blood sugar - уровень сахара в крови натощак) и chol (сыворотка холестеральная) слабо коррелируют с целевым признаком (0.03 и 0.09 соответственно), ввиду чего уберем данные признак из рассмотрения, чтобы предотвратить возможное ухудшение параметров работы моделей.

```
In [54]: data = data.drop(['fbs', 'chol'], axis=1)
```

```
In [55]: fig, ax = plt.subplots(figsize=(20,10))
         sns.heatmap(data.corr(), annot=True, fmt='.2f', ax=ax)
```

```
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x1580ace96d8>
```



3.4. Выбор метрик для оценки качества моделей

balanced_accuracy_score - сбалансированная точность в задачах двоичной и мультиклассовой классификации для решения проблемы несбалансированных наборов данных.

precision_score - доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

recall_score - доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

f1_score - объединяет precision и recall в единую метрику

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

```
In [56]: from sklearn.metrics import balanced_accuracy_score
         from sklearn.metrics import precision_score, recall_score
         from sklearn.metrics import f1_score
```

3.5. Выбор моделей для решения задачи классификации

SGDClassifier - стохастический градиентный спуск.
DecisionTreeClassifier - дерево решений.
RandomForestClassifier - случайный лес.

```
In [57]: from sklearn.linear_model import SGDClassifier
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier
```

3.6. Разделение выборки на обучающую и тестовую

```
In [58]: target = data_cleared['target']
         data_cleared = data_cleared.drop('target', axis=1)

In [59]: from sklearn.model_selection import train_test_split
         X_train, X_test, Y_train, Y_test = train_test_split(
             data_cleared,
             target,
             test_size=0.2,
             random_state=1
         )

In [60]: X_train.shape, Y_train.shape

Out[60]: ((242, 13), (242,))

In [61]: X_test.shape, Y_test.shape

Out[61]: ((61, 13), (61,))
```

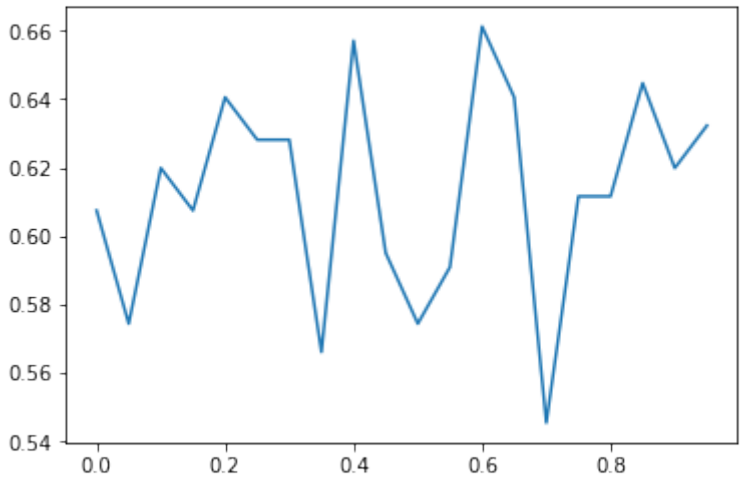
3.7. Построение базового решения (baseline) для выбранных моделей без подбора гиперпараметров

```
In [62]: sgd = SGDClassifier().fit(X_train, Y_train)
         predicted_sgd = sgd.predict(X_test)

In [63]: def print_accuracy_metrics(Y_test, predicted_sgd):
           print("balanced_accuracy_score {}".format(
               balanced_accuracy_score(Y_test, predicted_sgd)))
           print("precision_score {}".format(
               precision_score(Y_test, predicted_sgd, average='weighted')))
           print("recall_score {}".format(
               recall_score(Y_test, predicted_sgd, average='weighted')))
           print("f1_score {}".format(
               f1_score(Y_test, predicted_sgd, average='weighted')))

In [64]: print_accuracy_metrics(Y_test, predicted_sgd)

balanced_accuracy_score 0.689247311827957
precision_score 0.6908665105386416
recall_score 0.6885245901639344
f1_score 0.6880218049987431


In [65]: dt = DecisionTreeClassifier().fit(X_train, Y_train)
         predicted_dt = dt.predict(X_test)

In [66]: print_accuracy_metrics(Y_test, predicted_dt)

balanced_accuracy_score 0.7209677419354839
precision_score 0.7213998021481063
recall_score 0.7213114754098361
f1_score 0.7211615219395159


In [67]: rfc = RandomForestClassifier().fit(X_train, Y_train)
         predicted_rfc = rfc.predict(X_test)

In [68]: print_accuracy_metrics(Y_test, predicted_rfc)

balanced_accuracy_score 0.7365591397849462
precision_score 0.7413078724554134
recall_score 0.7377049180327869
f1_score 0.7362855723511461
```

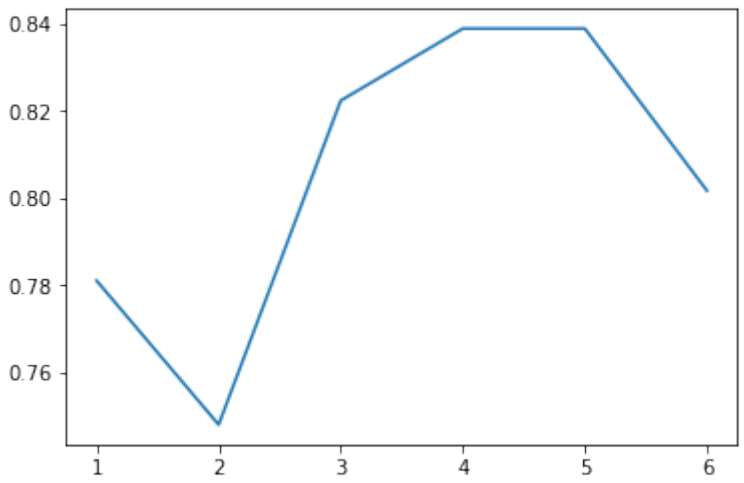
3.8. Подбор гиперпараметров для выбранных моделей

```
In [69]: from sklearn.model_selection import GridSearchCV

In [70]: n_range = np.array(range(0,100,5))
         n_range = n_range / 100
         tuned_parameters = [{'l1_ratio': n_range}]
         tuned_parameters
```

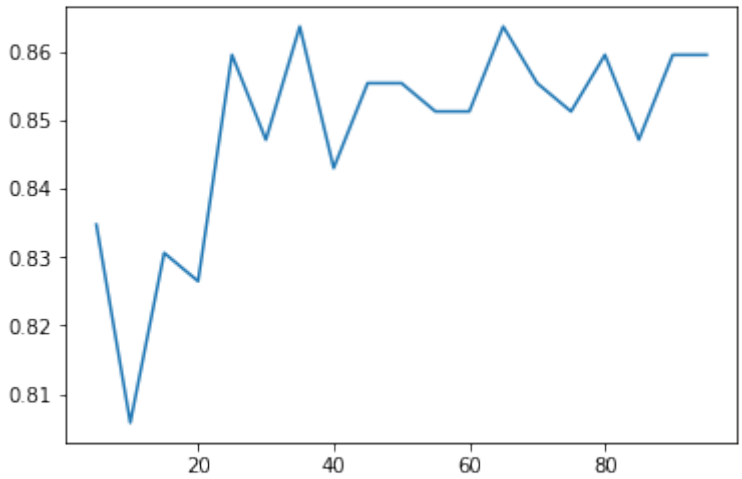
```
Out[70]: [{'l1_ratio': array([ 0.  ,  0.05,  0.1 ,  0.15,  0.2 ,  0.25,  0.3 ,
                  0.45,  0.5 ,  0.55,  0.6 ,  0.65,  0.7 ,  0.75,  0.8 ,  0.8
                  0.9 ,  0.95])}]

In [71]: clf_gs_sgd = GridSearchCV(SGDClassifier(), tuned_parameters, cv=5,
                                   scoring='accuracy')
         clf_gs_sgd.fit(X_train, Y_train)

Out[71]: GridSearchCV(cv=5, error_score='raise-deprecating',
                      estimator=SGDClassifier(alpha=0.0001, average=False,
                                              class_weight=None, early_stoppir
                                              epsilon=0.1, eta0=0.0, fit_inter
                                              l1_ratio=0.15, learning_rate='op
                                              loss='hinge', max_iter=1000,
                                              n_iter_no_change=5, n_jobs=None,
                                              penalty='l2', power_t=0.5,
                                              random_state=None, shuffle=True,
                                              validation_fraction=0.1, verbose
                                              warm_start=False),
                      iid='warn', n_jobs=None,
                      param_grid=[{'l1_ratio': array([ 0.  ,  0.05,  0.1 ,  0.
                  0.45,  0.5 ,  0.55,  0.6 ,  0.65,  0.7 ,  0.75,  0.8 ,  0.85,
                  0.9 ,  0.95])}],
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=
                      scoring='accuracy', verbose=0)

In [72]: clf_gs_sgd.best_params_

Out[72]: {'l1_ratio': 0.59999999999999998}

In [73]: plt.plot(n_range, clf_gs_sgd.cv_results_['mean_test_score'])

Out[73]: [<matplotlib.lines.Line2D at 0x1580aec2cc0>]
```



```
In [74]: n_range = np.array(range(1,7,1))
         tuned_parameters = [{'max_depth': n_range}]
         tuned_parameters

Out[74]: [{'max_depth': array([1, 2, 3, 4, 5, 6])}]

In [75]: clf_gs_dt = GridSearchCV(DecisionTreeClassifier(random_state=1), tune
                                  cv=5, scoring='accuracy')
         clf_gs_dt.fit(X_train, Y_train)

Out[75]: GridSearchCV(cv=5, error_score='raise-deprecating',
                      estimator=DecisionTreeClassifier(class_weight=None,
                                                       criterion='gini', max_d
                                                       max_features=None,
                                                       max_leaf_nodes=None,
                                                       min_impurity_decrease=0
                                                       min_impurity_split=None
                                                       min_samples_leaf=1,
                                                       min_samples_split=2,
                                                       min_weight_fraction_lea
                                                       presort=False, random_s
                                                       splitter='best'),
                      iid='warn', n_jobs=None,
                      param_grid=[{'max_depth': array([1, 2, 3, 4, 5, 6])}],
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=
                      scoring='accuracy', verbose=0)

In [76]: clf_gs_dt.best_params_

Out[76]: {'max_depth': 4}

In [77]: plt.plot(n_range, clf_gs_dt.cv_results_['mean_test_score'])

Out[77]: [<matplotlib.lines.Line2D at 0x1580b5d0630>]
```



```
In [78]: rfc_n_range = np.array(range(5,100,5))
         rfc_tuned_parameters = [{'n_estimators': rfc_n_range}]
         rfc_tuned_parameters

Out[78]: [{'n_estimators': array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55,
                  90, 95])}]

In [79]: gs_rfc = GridSearchCV(RandomForestClassifier(), rfc_tuned_parameters,
                               scoring='accuracy')
         gs_rfc.fit(X_train, Y_train)

Out[79]: GridSearchCV(cv=5, error_score='raise-deprecating',
                      estimator=RandomForestClassifier(bootstrap=True, class_w
                                                       criterion='gini', max_d
                                                       max_features='auto',
                                                       max_leaf_nodes=None,
                                                       min_impurity_decrease=0
                                                       min_impurity_split=None
                                                       min_samples_leaf=1,
                                                       min_samples_split=2,
                                                       min_weight_fraction_lea
                                                       n_estimators='warn', n_
                                                       oob_score=False,
                                                       random_state=None, verb
                                                       warm_start=False),
                      iid='warn', n_jobs=None,
                      param_grid=[{'n_estimators': array([ 5, 10, 15, 20, 25,
                 90, 95])}],
                      pre_dispatch='2*n_jobs', refit=True, return_train_score=
                      scoring='accuracy', verbose=0)

In [80]: gs_rfc.best_params_

Out[80]: {'n_estimators': 35}

In [81]: plt.plot(rfc_n_range, gs_rfc.cv_results_['mean_test_score'])

Out[81]: [<matplotlib.lines.Line2D at 0x1580b63c4a8>]
```



3.9. Оценка качества работы моделей с подобранными гиперпараметрами

```
In [82]: import warnings
         warnings.filterwarnings('ignore')

         sgd_optimized = SGDClassifier(l1_ratio=clf_gs_sgd.best_params_['l1_ra
         predicted_sgd_opt = sgd_optimized.predict(X_test)

In [83]: print_accuracy_metrics(Y_test, predicted_sgd)
         print()
         print_accuracy_metrics(Y_test, predicted_sgd_opt)

balanced_accuracy_score 0.689247311827957
precision_score 0.6908665105386416
recall_score 0.6885245901639344
f1_score 0.6880218049987431

balanced_accuracy_score 0.532258064516129
precision_score 0.7582661850514032
recall_score 0.5245901639344263
f1_score 0.39315237473277626


In [84]: dt_optimized = DecisionTreeClassifier(max_depth=clf_gs_dt.best_params
         predicted_dt_opt = dt_optimized.predict(X_test)

In [85]: print_accuracy_metrics(Y_test, predicted_dt)
         print()
         print_accuracy_metrics(Y_test, predicted_dt_opt)

balanced_accuracy_score 0.7209677419354839
precision_score 0.7213998021481063
recall_score 0.7213114754098361
f1_score 0.7211615219395159

balanced_accuracy_score 0.7543010752688173
precision_score 0.7545037898818968
recall_score 0.7540983606557377
f1_score 0.7540983606557377


In [86]: rfc_optimized = RandomForestClassifier(n_estimators=gs_rfc.best_param
         predicted_rfc_opt = rfc_optimized.predict(X_test)

In [87]: print_accuracy_metrics(Y_test, predicted_rfc)
         print()
         print_accuracy_metrics(Y_test, predicted_rfc_opt)

balanced_accuracy_score 0.7365591397849462
precision_score 0.7413078724554134
recall_score 0.7377049180327869
f1_score 0.7362855723511461
```

```
balanced_accuracy_score 0.7370967741935484
precision_score 0.7384500745156483
recall_score 0.7377049180327869
f1_score 0.7372809496890899
```

4. Выводы

Подбор гиперпараметров для выбранных моделей машинного обучения позволил увеличить точность решения задачи классификации на обучаемых моделях. Наибольший прирост в точности получила модель стохастического градиентного спуска. Однако наиболее точно с задачей классификации на данном датасете справляется дерево решений, как до подбора гиперпараметров, так и после.

5. Список литературы

1. Heart Disease UCI: https://www.kaggle.com/ronitf/heart-disease-uci
2. Model evaluation: quantifying the quality of predictions: https://scikit-learn.org/stable/modules/model_evaluation.html
3. Model selection: choosing estimators and their parameters: https://scikit-learn.org/stable/tutorial/statistical_inference/model_selection.html
4. SGDClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
5. DecisionTreeClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
6. RandomForestClassifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html