

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №41

ЗАЩИЩЕНА С ОЦЕНКОЙ
РУКОВОДИТЕЛЬ

К.т.н., доц.

Е.Л. Турнецкая

должность, уч. степень, звание

подпись, дата

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №4

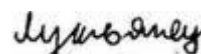
Применение методов классификации

по курсу: Методология и технология проектирования информационных
систем

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. №

М320М



П. Е. Лукьянец

подпись, дата

инициалы, фамилия

Санкт-Петербург 2023

Цель работы: изучить алгоритмы и методы классификации на практике.

Ход выполнения работы

Для работы был выбран третий датасет из списка под названием «3cancer». В данном датасете представлена информация о пациентах, болеющих раком: id, Толщина скопления, Однородность размера клеток, Однородность формы клеток, Краевая адгезия, Размер отдельных эпителиальных клеток, Голые ядра, Бледный хроматин, Нормальные ядрышки, Митозы, Класс (2 для доброкачественных, 4 для злокачественных). Чтобы начать работу, импортируем библиотеку, а затем считываем CSV файл. Выведем первые 20 строк с помощью метода `head`.. Часть данных этого датасета представлена на рисунке 1.



```
1 "id","clump_thickness","size_uniformity","shape_uniformity","marginal_adhesion","epithelial_size","bare_nucleoli","bland_chromatin","normal_nucleoli","mitoses","class"
2 1000025,5,1,1,1,2,1,3,1,1,2
3 1002945,5,4,4,5,7,10,3,2,1,2
4 1015425,3,1,1,1,2,2,3,1,1,2
5 1016277,6,8,8,1,3,4,3,7,1,2
6 1017023,4,1,1,3,2,1,3,1,1,2
7 1017122,8,10,10,8,7,10,9,7,1,4
8 1018099,1,1,1,1,2,10,3,1,1,2
9 1018561,2,1,2,1,2,1,3,1,1,2
10 1033078,2,1,1,1,2,1,1,1,5,2
11 1033078,4,2,1,1,2,1,2,1,1,2
12 1035283,1,1,1,1,1,3,1,1,2
13 1036172,2,1,1,1,2,1,2,1,1,2
14 1041801,5,3,3,3,2,3,4,4,1,4
15 1043999,1,1,1,1,2,3,3,1,1,2
16 1044572,8,7,5,10,7,9,5,5,4,4
17 1047630,7,4,6,4,6,1,4,3,1,4
18 1048672,4,1,1,1,2,1,2,1,1,2
19 1049815,4,1,1,1,2,1,3,1,1,2
20 1050670,10,7,7,6,4,10,4,1,2,4
21 1050718,6,1,1,1,2,1,3,1,1,2
22 1054590,7,3,2,10,5,10,5,4,4,4
23 1054593,10,5,5,3,6,7,7,10,1,4
24 1056784,3,1,1,1,2,1,2,1,1,2
```

Рисунок 1 – используемый датасет

Для работы с ним использовалась библиотека Pandas.

Работа была выполнена при помощи Visual Studio Code, а также Jupyter Notebook.

Ссылка на GitHub репозиторий с файлами:
<https://github.com/NinjaCaratist/MTPIS>

Чтобы начать работу, импортируем библиотеку, а затем считываем CSV файл.

```
import pandas as pd

df = pd.read_csv("3cancer.csv")
```

Рисунок 2 – скриншот кода

Выведем первые 20 строк с помощью метода head. На рисунке 3 показан код, а на рисунке 4 – результат его работы.

```
#2
print(df.head(20))
```

Рисунок 3 – скриншот кода

	id	clump_thickness	size_uniformity	shape_uniformity	\
0	1000025	5	1	1	
1	1002945	5	4	4	
2	1015425	3	1	1	
3	1016277	6	8	8	
4	1017023	4	1	1	
5	1017122	8	10	10	
6	1018099	1	1	1	
7	1018561	2	1	2	
8	1033078	2	1	1	
9	1033078	4	2	1	
10	1035283	1	1	1	
11	1036172	2	1	1	
12	1041801	5	3	3	
13	1043999	1	1	1	
14	1044572	8	7	5	
15	1047630	7	4	6	
16	1048672	4	1	1	
17	1049815	4	1	1	
18	1050670	10	7	7	
19	1050718	6	1	1	

	marginal_adhesion	epithelial_size	bare_nucleoli	bland_chromatin
0	1	2	1	3
1	5	7	10	3
2	1	2	2	3
3	1	3	4	3
4	3	2	1	3
5	8	7	10	9
6	1	2	10	3
7	1	2	1	3
8	1	2	1	1
9	1	2	1	2
10	1	1	1	3
11	1	2	1	2
12	3	2	3	4
13	1	2	3	3
14	10	7	9	5
15	4	6	1	4
16	1	2	1	2
17	1	2	1	3
18	6	4	10	4
19	1	2	1	3

	normal_nucleoli	mitoses	class
0	1	1	2
1	2	1	2
2	1	1	2
3	7	1	2
4	1	1	2
5	7	1	4
6	1	1	2
7	1	1	2
8	1	5	2
9	1	1	2
10	1	1	2
11	1	1	2
12	4	1	4
13	1	1	2
14	5	4	4
15	3	1	4
16	1	1	2
17	1	1	2
18	1	2	4
19	1	1	2

Рисунок 4 – результат вывода

Как уже было сказано, данная таблица содержит информацию об анализах пациентов с раком. Предметная область – медицина.

Опишем колонки подробнее:

1. id
2. Толщина скопления: 1–10
3. Однородность размера клеток: 1–10
4. Однородность формы клеток: 1–10
5. Краевая адгезия: 1–10
6. Размер отдельных эпителиальных клеток: 1 - 10
7. Голые ядра: 1 - 10
8. Бледный хроматин: 1 - 10
9. Нормальные ядрышки: 1 - 10
10. Митозы: 1 - 10
11. Класс: (2 для доброкачественных, 4 для злокачественных)

Теперь с помощью метода «.info» оценим данные. Этот метод возвращает название столбцов, типы данных, количество ненулевых объектов каждом столбце. Этот метод возвращает название столбцов, типы данных, количество ненулевых объектов каждом столбце. Результат работы метода представлен на рисунке 5.

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 699 entries, 0 to 698
Data columns (total 11 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   id                                     699 non-null    int64
1   clump_thickness                       699 non-null    int64
2   size_uniformity                      699 non-null    int64
3   shape_uniformity                     699 non-null    int64
4   marginal_adhesion                    699 non-null    int64
5   epithelial_size                      699 non-null    int64
6   bare_nucleoli                        699 non-null    object
7   bland_chromatin                      699 non-null    int64
8   normal_nucleoli                      699 non-null    int64
9   mitoses                             699 non-null    int64
10  class                                699 non-null    int64
dtypes: int64(10), object(1)
memory usage: 60.2+ KB

```

Рисунок 5 – результат вывода

Теперь выведем на экран названия столбцов с помощью `df.columns`.
Названия всех колонок приемлемы.

```
print(df.columns)
```

Рисунок 6 – скриншот кода

```

Index(['id', 'clump_thickness', 'size_uniformity', 'shape_uniformity',
      'marginal_adhesion', 'epithelial_size', 'bare_nucleoli',
      'bland_chromatin', 'normal_nucleoli', 'mitoses', 'class'],
      dtype='object')

```

Рисунок 7 – результат вывода

Найдём пропуски и устраним их. При помощи метода «`isna`» найдём все пропуски в таблице, а также при помощи `sum` выведем количество пропусков в каждом столбце. Пропусков в данном наборе данных нет. Код представлен на рисунке 8.

```
print(df.isna().sum())
```

Рисунок 8 – скриншот кода

```
id                0
clump_thickness   0
size_uniformity   0
shape_uniformity  0
marginal_adhesion 0
epithelial_size   0
bare_nucleoli     0
bland_chromatin   0
normal_nucleoli   0
mitoses           0
class             0
dtype: int64
```

Рисунок 9 – результат вывода

Проверим данные на наличие дубликатов. Удалим дубликаты, при помощи метода "drop_duplicates", а также перераспределим индексы. Также выведем уникальные значения каждого столбца. Столбец 'bare_nucleoli' содержит некорректные значения '?'. Заменяем их при помощи метода 'replace' на NaN, и выкинем их.

```

import numpy as np

print("Количесвто дубликатов: " + str(df.duplicated().sum()))
df = df.drop_duplicates().reset_index()
print("Количесвто дубликатов: " + str(df.duplicated().sum()))

print("Unique id")
print(df['id'].unique())
print("Unique clump_thickness")
print(df['clump_thickness'].unique())
print("Unique size_uniformity")
print(df['size_uniformity'].unique())
print("Unique shape_uniformity")
print(df['shape_uniformity'].unique())
print("Unique marginal_adhesion")
print(df['marginal_adhesion'].unique())
print("Unique epithelial_size")
print(df['epithelial_size'].unique())
print("Unique bare_nucleoli")
print(df['bare_nucleoli'].unique())
print("Unique bland_chromatin")
print(df['bland_chromatin'].unique())
print("Unique normal_nucleoli")
print(df['normal_nucleoli'].unique())
print("Unique mitoses")
print(df['mitoses'].unique())
print("Unique class ")
print(df['class'].unique())

df['bare_nucleoli'] = df['bare_nucleoli'].replace('?', np.NaN)
df = df.dropna(subset=['bare_nucleoli']).reset_index(drop=True)
print("Unique bare_nucleoli")
print(df['bare_nucleoli'].unique())

```

Рисунок 10 – скриншот кода

```

Количесвто дубликатов: 8
Количесвто дубликатов: 0

```



```

1355260 1365075 1365328 1368267 1368273 1368882 1369821 1371026
1371920 466906 534555 536708 566346 603148 654546 714039
763235 776715 841769 888820 897471]
Unique clump_thickness
[ 5 3 6 4 8 1 2 7 10 9]
Unique size_uniformity
[ 1 4 8 10 2 3 7 5 6 9]
Unique shape_uniformity
[ 1 4 8 10 2 3 5 6 7 9]
Unique marginal_adhesion
[ 1 5 3 8 10 4 6 2 9 7]
Unique epithelial_size
[ 2 7 3 1 6 4 5 8 10 9]
Unique bare_nucleoli
['1' '10' '2' '4' '3' '9' '7' '?' '5' '8' '6']
Unique bland_chromatin
[ 3 9 1 2 4 5 7 8 6 10]
Unique normal_nucleoli
[ 1 2 7 4 5 3 10 6 9 8]
Unique mitoses
[ 1 5 4 2 3 7 10 8 6]
Unique class
[2 4]
Unique bare_nucleoli
['1' '10' '2' '4' '3' '9' '7' '5' '8' '6']

```

Рисунок 11 – результат вывода

Проверим все ли типы данных соответствуют действительности. Все столбцы, кроме Голых ядер соответствуют своему типу. Поэтому при помощи метода «to_numeric» изменяем тип на int. Выведем информацию.

```

df['bare_nucleoli'] = pd.to_numeric(df['bare_nucleoli'])
df.info()

0.0s

```

Рисунок 12 – скриншот кода

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 675 entries, 0 to 674
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  -
0   index                  675 non-null    int64
1   id                     675 non-null    int64
2   clump_thickness        675 non-null    int64
3   size_uniformity        675 non-null    int64
4   shape_uniformity       675 non-null    int64
5   marginal_adhesion     675 non-null    int64
6   epithelial_size       675 non-null    int64
7   bare_nucleoli         675 non-null    int64
8   bland_chromatin       675 non-null    int64
9   normal_nucleoli       675 non-null    int64
10  mitoses               675 non-null    int64
11  class                 675 non-null    int64
dtypes: int64(12)
memory usage: 63.4 KB

```

Рисунок 13 – результат вывода

Прежде чем отправить данные на вход модели и получить прогнозы, проведём — EDA, или исследовательский анализ данных. На этом этапе изучим распределения отдельных признаков и целевой переменной. Построим графики методом `kdeplot()`.

Также построим тепловую карту корреляции, чтобы увидеть взаимосвязь между значениями и целевым признаком.

Удалим атрибут "id", так как он не несёт смысловой нагрузки для модели.

```

import seaborn as sb
import matplotlib.pyplot as plt
sb.kdeplot(df['clump_thickness'],vertical=True,color='blue',shade=False, label='clump_thickness')
sb.kdeplot(df['size_uniformity'],vertical=True,color='red',shade=False)
sb.kdeplot(df['shape_uniformity'],vertical=True,color='green',shade=False)
sb.kdeplot(df['marginal_adhesion'],vertical=True,color='yellow',shade=False)
sb.kdeplot(df['epithelial_size'],vertical=True,color='cyan',shade=False)
sb.kdeplot(df['bare_nucleoli'],vertical=True,color='orange',shade=False)
sb.kdeplot(df['bland_chromatin'],vertical=True,color='violet',shade=False)
sb.kdeplot(df['normal_nucleoli'],vertical=True,color='pink',shade=False)
sb.kdeplot(df['mitoses'],vertical=True,color='black',shade=False)
sb.kdeplot(df['class'],vertical=True,color='olive',shade=False)
plt.show()

corr = df.corr(method = 'pearson')
sb.heatmap(corr, cmap="Blues", annot=True)
plt.show()

df = df.drop('id', axis = 1)

```

Рисунок 14 – скриншот кода

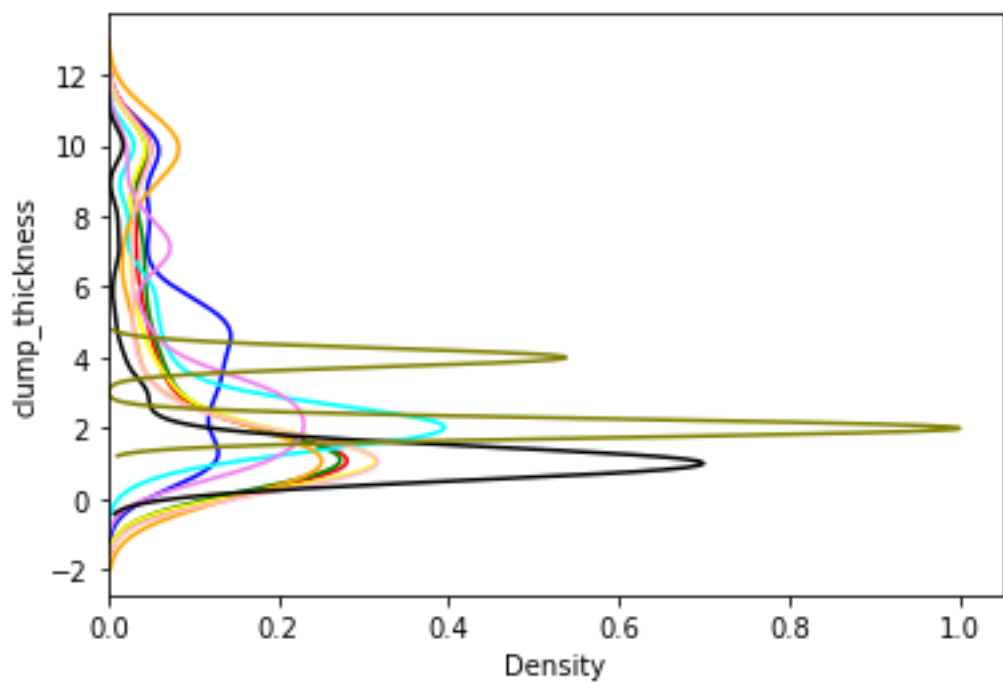


Рисунок 15 – построенный график

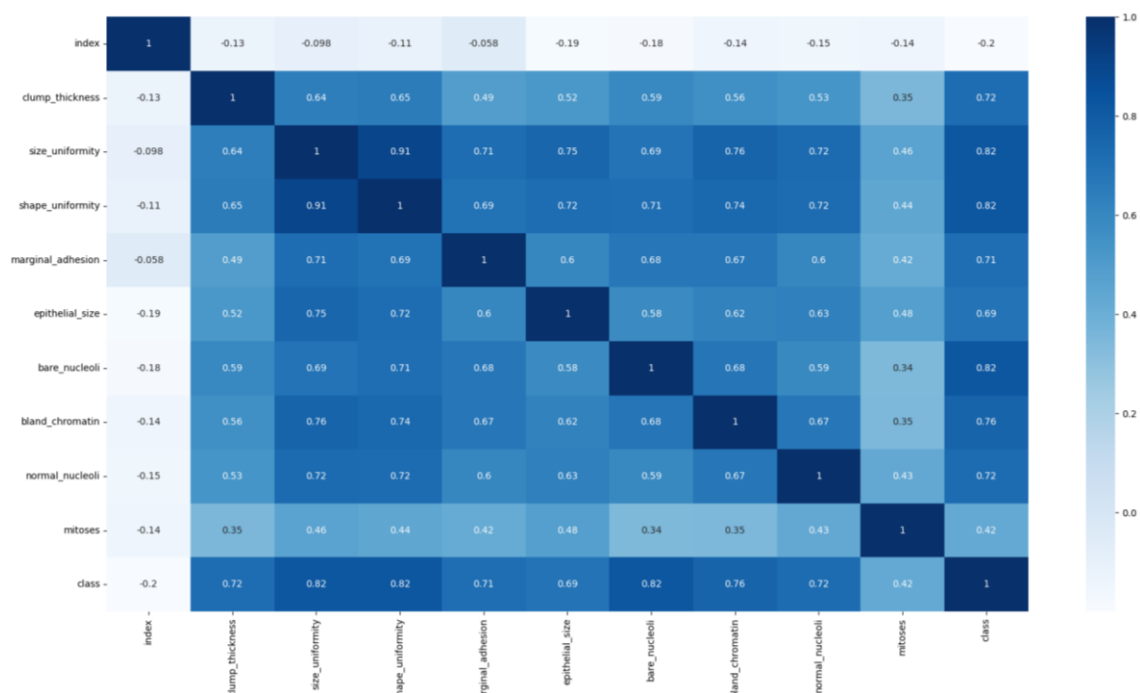


Рисунок 16 – построенная тепловая карта

Перед тем как разделять данные на тестовую и обучающую выборки и обучать на них различные модели, пропишем функцию проверки обученной модели, чтобы затем было проще её вызывать.

Матрица ошибок

Возможные значения классов: 2 и 4. Модель тоже может выдавать итоговый прогноз в виде значения одного из двух классов. Тогда для каждого объекта прогноз относится к одной из четырех групп:

- 1) Прогноз модели = 1, реальное значение = 1. Такие прогнозы называют True Positive («истинно положительные») — сокращённо TP.
- 2) Прогноз модели = 1, реальное значение = 0. Такие прогнозы называют False Positive («ложно положительные») — сокращённо FP.
- 3) Прогноз модели = 0, реальное значение = 1. Такие прогнозы называют False Negative («ложно отрицательные») — сокращённо FN.
- 4) Прогноз модели = 0, реальное значение = 0. Такие прогнозы называют True Negative («истинно отрицательные») — сокращённо TN

Доля правильных ответов

Это доля верно угаданных ответов из всех прогнозов. Чем ближе значение accuracy к 100%, тем лучше. Метрику рассчитывают функцией `accuracy_score` из модуля `metrics`. На вход функция принимает верные и спрогнозированные значения классов на валидационной выборке.

Точность (precision) и полнота (recall)

Чтобы оценить модель без привязки к соотношению классов, рассчитывают эти метрики.

`Precision` говорит, какая доля прогнозов относительно "1" класса верна. То есть смотрим долю правильных ответов только среди целевого класса.

Вторая метрика нацелена на минимизацию противоположных рисков — `recall` показывает, сколько реальных объектов "1" класса вы смогли обнаружить с помощью модели.

Каждая метрика принимает значения от 0 до 1. Чем ближе к единице, тем лучше. Однако при настройке параметров модели — обычно порога вероятности, после которого мы относим объект к классу "1" — оптимизация одной метрики часто приводит к ухудшению другой. Метрики точности и полноты также реализованы в модуле `metrics` в функциях `"precision_score"` и `"recall_score"`.

F1-мера

Так как `precision` и `recall` направлены на избежание противоположных рисков, нужна сводная метрика, учитывающая баланс между метриками. В `sklearn.metrics` F1-меру вычисляют методом `"f1_score"`.

ROC - кривая

Для оценки качества классификатора (модели классификации) применяют метрику `roc_auc`, или площадь под кривой ошибок — AUC-ROC. Для оценки качества классификатора (модели классификации) применяют метрику `roc_auc`, или площадь под кривой ошибок — AUC-ROC.

```

from sklearn.metrics import confusion_matrix, precision_score, recall_score, f1_score
from sklearn import metrics
from sklearn.metrics import accuracy_score, roc_auc_score
def ROC_curve (test_y,roc_auc,rf_probs):
    test_y = test_y.replace(to_replace = 2, value = 0)
    test_y = test_y.replace(to_replace = 4, value = 1)
    print (test_y)
    fpr, tpr, _ = metrics.roc_curve (test_y, rf_probs)

    #create ROC curve
    plt.plot (fpr,tpr,label=" AUC= "+str(roc_auc))
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()

def model_check (test_y,predictions,rf_probs):
    # Матрица ошибок
    cm = confusion_matrix(test_y,predictions)
    tn, fp, fn, tp = cm.ravel() # "выпрямляем" матрицу, чтобы вытащить нужные значения
    print("Матрица ошибок:")
    print("TN:" + str(tn), "FP:" + str(fp), "FN:" + str(fn), "TP:" + str(tp))

    #Доля правильных ответов
    acc = accuracy_score(test_y,predictions)
    print("Accuracy score = " + str(acc))

    #Точность (precision) и полнота (recall)
    precision = precision_score (test_y,predictions,pos_label=2)
    recall = recall_score (test_y,predictions,pos_label=2)
    print("precision_score = " + str(precision))
    print("recall_score = " + str(recall))

    #F1-мера
    f1= f1_score(test_y,predictions, pos_label=2)
    print("F1 score = " + str(f1))

    #ROC кривая
    roc_auc = roc_auc_score(test_y,rf_probs)
    print("ROC = " + str(roc_auc))
    ROC_curve (test_y,roc_auc,rf_probs)

```

Рисунок 17 – скриншот кода

Теперь разделим набор данных на тестовую и обучающую выборки соотношением 3 к 7.

Произведём обучение и прогнозирование методом fit-predict. Каждой модели в sklearn соответствует отдельная структура данных. DecisionTreeClassifier (англ. «классификатор дерева решений») — это структура данных для классификации деревом решений.

В переменной `model` будет храниться модель. Что обучить модель, нужно запустить алгоритм обучения.

Разделим датафрейм на столбец с целевой переменной (`class`) и остальные данные. Теперь уже можно построить взаимосвязь и на её основании спрогнозировать `y` по новым `X`. Чтобы запустить обучение, вызовем метод `fit()` и передадим ему как параметр данные.

Чтобы построить прогнозы для набора данных, хватит одной строчки кода и вызова метода `predict()`. У нас осталась отложенная порция данных, для которых мы знаем признаки и ответы. На этом этапе мы берём только признаки, передаём их на вход обученной модели и сохраняем предсказанные значения.

Теперь вызываем функцию оценки качества и смотрим на оценки и график.

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.3)

train.info()
test.info()
model = DecisionTreeClassifier()

train_y = train["class"]
print(train_y)
train_X = train.drop('class', axis = 1)

train_X.info()

test_y = test["class"]
test_X = test.drop('class', axis = 1)
test.info()

model.fit(train_X,train_y)
predictions = model.predict(test_X)
print("predictions:")
print(predictions)
💡
rf_probs = model.predict_proba(test_X)[::, 1]

model_check (test_y,predictions,rf_probs)

```

Рисунок 18 – скриншот кода


```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 472 entries, 589 to 300
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   index                 472 non-null   int64
1   id                    472 non-null   int64
2   clump_thickness       472 non-null   int64
3   size_uniformity       472 non-null   int64
4   shape_uniformity      472 non-null   int64
5   marginal_adhesion     472 non-null   int64
6   epithelial_size       472 non-null   int64
7   bare_nucleoli         472 non-null   int64
8   bland_chromatin       472 non-null   int64
9   normal_nucleoli       472 non-null   int64
10  mitoses               472 non-null   int64
11  class                 472 non-null   int64
dtypes: int64(12)
memory usage: 47.9 KB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 203 entries, 96 to 54
Data columns (total 12 columns):
#   Column                Non-Null Count  Dtype
---  ---
0   index                 203 non-null   int64
1   id                    203 non-null   int64
2   clump_thickness       203 non-null   int64
3   size_uniformity       203 non-null   int64
4   shape_uniformity      203 non-null   int64
5   marginal_adhesion     203 non-null   int64
6   epithelial_size       203 non-null   int64
7   bare_nucleoli         203 non-null   int64
8   bland_chromatin       203 non-null   int64
9   normal_nucleoli       203 non-null   int64
10  mitoses               203 non-null   int64
11  class                 203 non-null   int64
dtypes: int64(12)
memory usage: 20.6 KB

```

```

predictions:
[4 4 2 2 4 2 2 4 2 4 2 2 4 2 4 4 4 2 2 2 2 4 2 4 4 2 4 2 2 4 4 2 2 2 2 4 2
 2 2 2 2 4 4 2 2 4 2 2 4 4 2 2 2 2 4 4 2 4 2 2 4 2 2 2 2 2 2 4 2 2 2 2 2 2
 4 2 4 4 2 4 4 4 2 4 2 4 4 4 2 2 2 2 4 2 4 2 2 2 2 2 2 2 2 2 2 4 4 2 2 2 2
 4 4 2 2 4 2 2 2 2 2 2 4 4 4 2 4 4 2 4 4 2 4 2 2 2 2 2 2 2 2 4 2 2 2 4 4 4
 4 4 2 4 2 4 2 2 2 2 2 4 4 2 4 2 4 2 2 2 2 2 4 2 2 2 2 2 2 2 2 2 4 2 4 4
 2 2 2 2 4 4 2 4 2 4 2 2 2 2 2 2 2 2 4]
Матрица ошибок:
TN:126 FP:5 FN:5 TP:67
Accuracy score = 0.9507389162561576
precision_score = 0.9618320610687023
recall_score = 0.9618320610687023
F1 score = 0.9618320610687023
ROC = 0.9461938083121291
96      1
258     1
669     0
487     0
84      1
      ..
474     0
131     0
477     0
157     0
54      1
Name: class, Length: 203, dtype: int64

```

Рисунок 19 – результат вывода

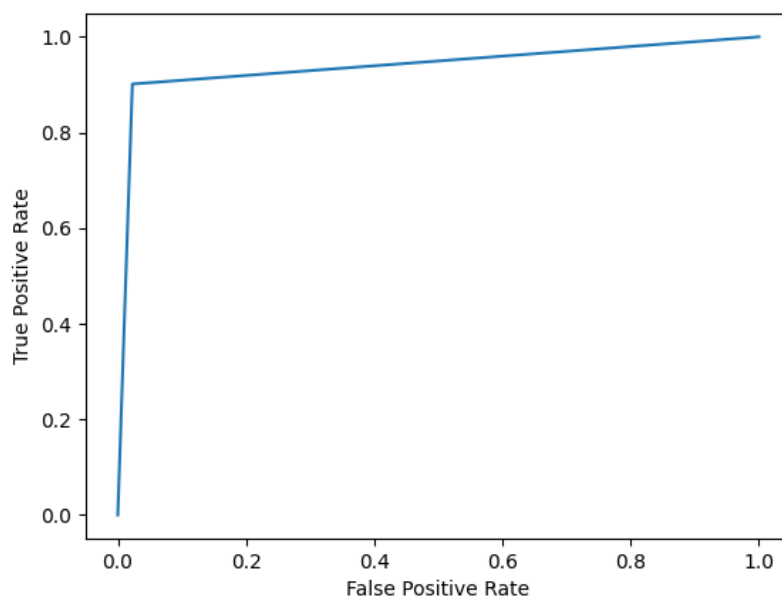


Рисунок 20 – построенный график

Снова разделим набор данных на тестовую и обучающую выборки соотношением 3 к 7.

Теперь произведём обучение и прогнозирование методом случайного леса - RandomForestClassifier.

В переменной clf будет храниться модель. Что обучить модель, нужно запустить алгоритм обучения.

Разделим датафрейм на столбец с целевой переменной (class) и остальные данные. Теперь уже можно построить взаимосвязь и на её основании спрогнозировать y по новым X. Чтобы запустить обучение, вызовем метод fit() и передадим ему как параметр данные.

Чтобы построить прогнозы для набора данных, хватит одной строчки кода и вызова метода predict(). У нас осталась отложенная порция данных, для которых мы знаем признаки и ответы. На этом этапе мы берём только признаки, передаём их на вход обученной модели и сохраняем предсказанные значения.

Теперь вызываем функцию оценки качества и смотрим на оценки и график.

```
from sklearn.ensemble import RandomForestClassifier
train, test = train_test_split(df, test_size=0.3)
train_y = train["class"]
train_X = train.drop('class', axis = 1)
test_y = test["class"]
test_X = test.drop('class', axis = 1)

clf = RandomForestClassifier(n_estimators=100)
clf.fit(train_X, train_y)
y_pred = clf.predict(test_X)
# Вероятности для каждого класса
rf_probs = clf.predict_proba(test_X)[::, 1]

model_check (test_y,y_pred,rf_probs)
```

Рисунок 21 – скриншот кода

```
Матрица ошибок:  
TN:123 FP:4 FN:3 TP:73  
Accuracy score = 0.9655172413793104  
precision_score = 0.9761904761904762  
recall_score = 0.968503937007874  
F1 score = 0.9723320158102767  
ROC = 0.9953895565685869  
323    0  
528    1  
493    1  
396    0  
182    1  
      ..  
375    0  
346    1  
471    0  
249    1  
185    1  
Name: class, Length: 203, dtype: int64
```

Рисунок 22 – результат вывода

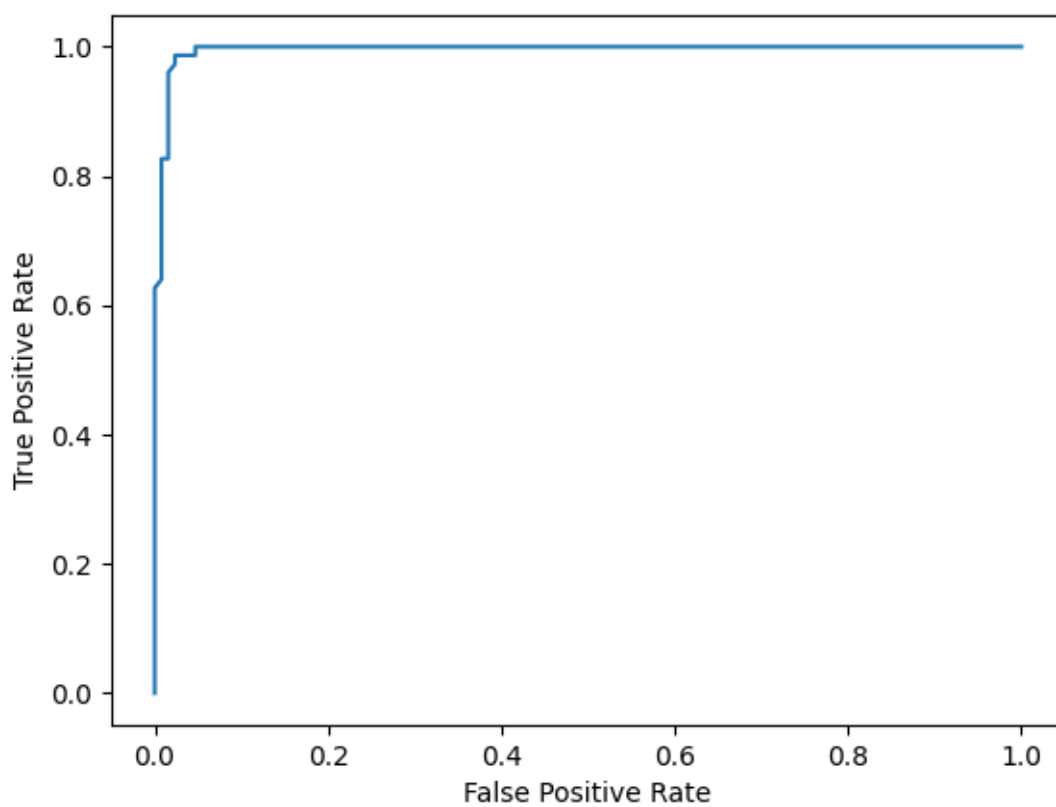


Рисунок 23 – построенный график

Вывод: Таким образом, в ходе выполнения лабораторной работы был выбран и описан выбранный датасет про пациентов с болезнью сердца, изучен интерфейс и возможности Jupyter Notebook, изучены базовые функции библиотеки Pandas и разработана программа, которая считывает данные, выводит о них информацию, удаляет дубликаты, пропуски, изменяет тип данных. Также были изучены методы классификации, стандартизированы данные, обучены различные модели, вычислены метрики качества, получена матрица неточностей и построен график ROC-кривой.