|  |  |
| --- | --- |
|  | 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 \_\_\_\_\_\_\_\_\_\_\_СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ (ИУ5)\_\_\_\_

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

по дисциплине \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Технологии машинного обучения \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

по теме \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ «Типовое исследование» \_\_\_ \_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Студент ИУ5Ц-82Б \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С. А. Павлов

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ю. Е. Гапанюк

(Подпись, дата) (И.О.Фамилия)

2024 г.

**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)**

УТВЕРЖДАЮ

Заведующий кафедрой \_\_\_ИУ5\_\_\_\_

(Индекс)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_В.И. Терехов\_\_

(И.О.Фамилия)

«\_07\_» \_\_\_\_февраля\_\_\_\_\_\_\_ 2024 г.

**ЗАДАНИЕ**

**на выполнение научно-исследовательской работы**

по теме «Типовое исследование»

Студент группы \_ИУ5Ц-82Б\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Павлов Сергей Алексеевич\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(Фамилия, имя, отчество)

Направленность НИР (учебная, исследовательская, практическая, производственная, др.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ИССЛЕДОВАТЕЛЬСКАЯ\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Источник тематики (кафедра, предприятие, НИР) \_\_\_\_\_КАФЕДРА\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

График выполнения НИР: 25% к \_\_\_\_ нед., 50% к \_\_\_\_ нед., 75% к \_\_\_ нед., 100% к \_\_\_\_ нед.

***Техническое задание***

1. Выбрать набор данных (датасет), подходящий для решения задачи классификации или задачи регрессии.

2. Для выбранного датасета на основе материалов лекции решить следующие задачи:

1. Поиск набора данных для построения моделей машинного обучения.

2. Разведочный анализ данных и заполнение пропусков.

3. Выбор признаков, их кодирование, масштабирование и формирование дополнительных признаков.

4. Корреляционный анализ данных и формирование выводов о построении моделей.

5. Выбор метрик для оценки качества моделей.

6. Выбор моделей для классификации.

7. Формирование обучающей и тестовой выборок.

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

9. Подбор гиперпараметров с использованием методов кросс-валидации.

10. Проверка качества моделей с оптимальными гиперпараметрами.

11. Формирование выводов о качестве моделей и представление результатов в виде графиков и текстовых описаний.

12. Создание веб-приложения для демонстрации модели машинного обучения с возможностью изменения гиперпараметров.

***Оформление научно-исследовательской работы:***

Расчетно-пояснительная записка на 24 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_07\_» \_\_\_\_февраля\_\_\_\_\_\_\_ 2024 г.

**Руководитель НИР**  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_Ю.Е. Гапанюк\_\_\_\_\_\_

(Подпись, дата) (И.О.Фамилия)

**Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_**С. А. Павлов**\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

# **Выполнение работы**

# **1. Поиск набора данных для построения моделей машинного обучения.**

Загрузим необходимые библиотеки. Также пропишем в тексте программы команду, позволяющую отображать графики в ячейках блокнота, и установим стиль графиков для отображения делений на осях графиков:

pip install scikit-learn

import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

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\_report

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, mean\_squared\_log\_error, median\_absolute\_error, r2\_score

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\_graphviz

from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor

from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor

from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor

%matplotlib inline

sns.set(style="ticks")

Загрузим обучающую выборку:

original\_train = pd.read\_csv('heart.csv', sep=",")

Сделаем дупликат выборки:

train = original\_train.drop\_duplicates()

# **2. Разведочный анализ данных и заполнение пропусков.**

Отобразим первые 5 строк датасета:

train.head()

Результат работы команды представлен на рисунке 1.

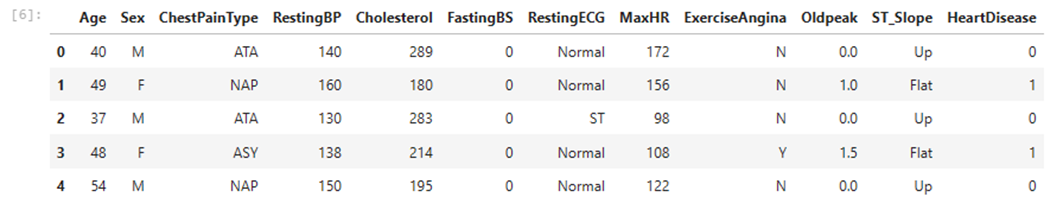


Рисунок 1 — Результат выполнения команды

Отобразим размер обучающего датасета:

train.shape

Результат работы команды представлен на рисунке 2.



Рисунок 2 — Результат выполнения команды

Представим список колонок с помощью команды:

train.columns

Результат работы команды представлен на рисунке 3.

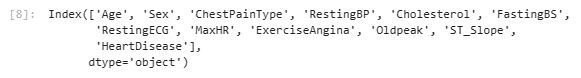


Рисунок 3 — Результат выполнения команды

Отобразим список колонок с типами данных:

train.dtypes

Результат работы команды представлен на рисунке 4.

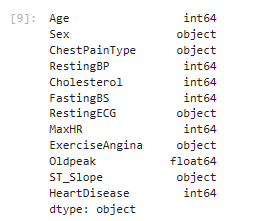


Рисунок 4 — Результат выполнения команды

Проверим, есть ли пропущенные значения:

train.isnull().sum()

Результат работы команды представлен на рисунке 5.

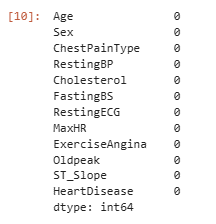


Рисунок 5 — Результат выполнения команды

Построение графиков для понимания структуры данных

Отобразим парные диаграммы:

sns.pairplot(train)

Результат работы команды представлен на рисунке 6.

Отобразим парные диаграммы с окрашиванием целевого признака:

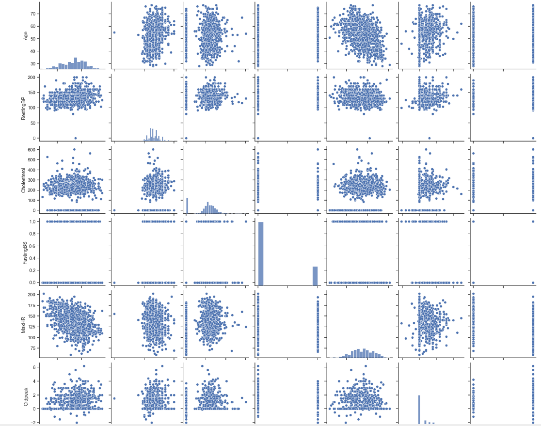


Рисунок 6 — Результат выполнения команды

sns.pairplot(train, hue="HeartDisease")

Результат работы команды представлен на рисунке 7.

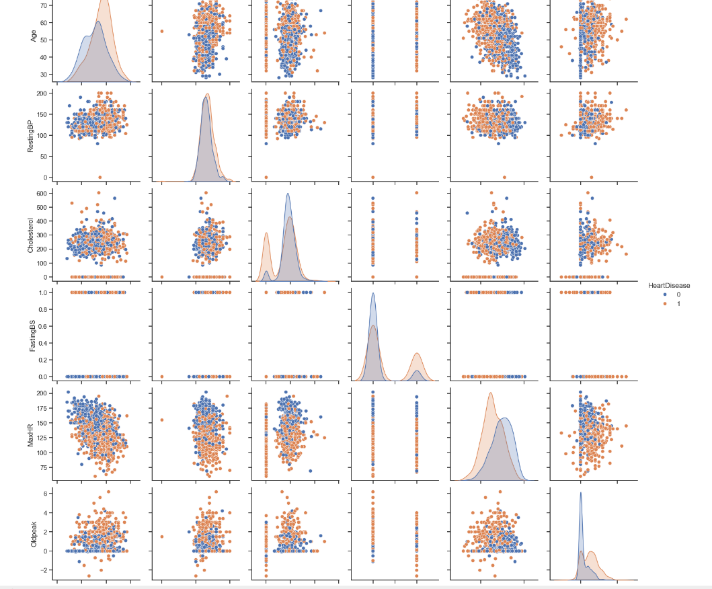


Рисунок 7— Результат выполнения команды

Убедимся, что целевой признак для задачи бинарной классификации содержит только 0 и 1:

train['HeartDisease'].unique()

Результат работы команды представлен на рисунке 8.



Рисунок 8 — Результат выполнения команды

Оценим дисбаланс классов для 'HeartDisease':

fig, ax = plt.subplots(figsize=(2,2))

plt.hist(train['HeartDisease'])

plt.show()

Результат работы команды представлен на рисунке 9.

Подсчитаем количество уникальных значений в столбце 'HeartDisease':

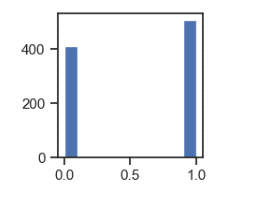


Рисунок 9 — Результат выполнения команды

train['HeartDisease'].value\_counts()

Результат работы команды представлен на рисунке 10.



Рисунок 10 — Результат выполнения команды

Подсчитаем дисбаланс классов:

total = train.shape[0]

class\_0, class\_1 = train['HeartDisease'].value\_counts()

print('Класс 0 составляет {}%, а класс 1 составляет {}%.'

.format(round(class\_0 / total, 4)\*100, round(class\_1 / total, 4)\*100))

Результат работы команды представлен на рисунке 11.



Рисунок 11 — Результат выполнения команды

Составим скрипичные диаграммы для числовых колонок:

for col in ['Age', 'RestingBP', 'Cholesterol', 'MaxHR']:

sns.violinplot(x=train[col])

plt.show()

Результат работы команды представлен на рисунке 12.

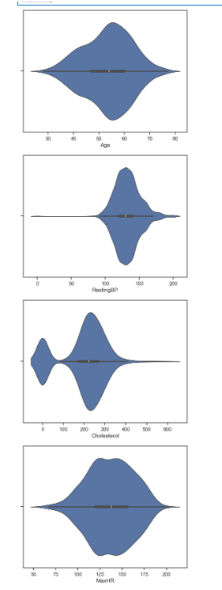


Рисунок 12— Результат выполнения команды

**3. Выбор признаков, их кодирование, масштабирование и формирование дополнительных признаков.**

Создадим вспомогательные колонки, чтобы наборы данных можно было разделить:

train = train.copy()

train['dataset'] = 'TRAIN'

Колонки для объединения:

join\_cols = ['Age', 'Sex', 'ChestPainType', 'RestingBP', 'Cholesterol', 'FastingBS',

'RestingECG', 'MaxHR', 'ExerciseAngina', 'Oldpeak', 'ST\_Slope',

'HeartDisease', 'dataset']

data\_all = pd.concat([train[join\_cols]])

Проверим корректность объединения:

assert data\_all.shape[0] == train.shape[0]

Отобразим первые 5 строк:

data\_all.head()

Результат работы команды представлен на рисунке 13.

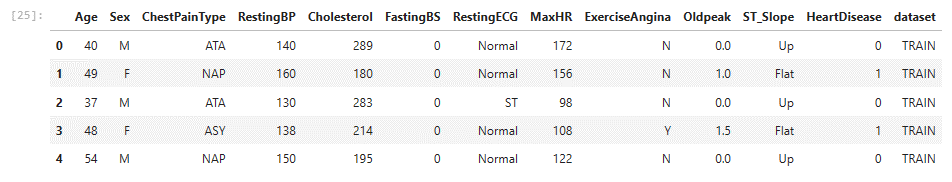


Рисунок 13 — Результат выполнения команды

Числовые колонки для масштабирования:

scale\_cols = ['Age', 'RestingBP', 'Cholesterol', 'FastingBS', 'MaxHR', 'Oldpeak']

sc1 = MinMaxScaler()

sc1\_data = sc1.fit\_transform(data\_all[scale\_cols])

Добавим масштабированные данные в набор данных:

for i in range(len(scale\_cols)):

col = scale\_cols[i]

new\_col\_name = col + '\_scaled'

data\_all[new\_col\_name] = sc1\_data[:,i]

Отобразим первые 5 строк:

data\_all.head()

Результат работы команды представлен на рисунке 14.

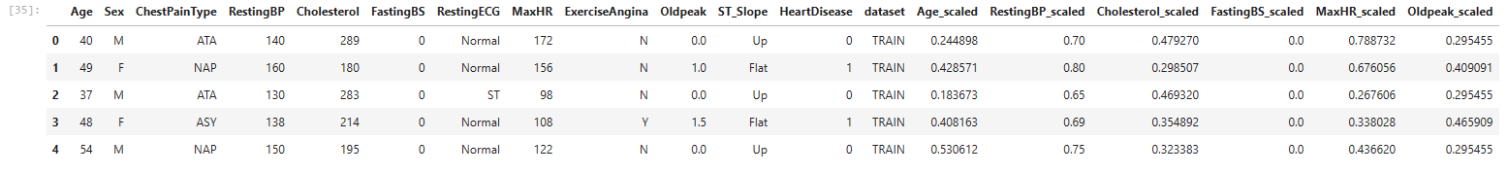


Рисунок 14 — Результат выполнения команды

Проверим, что масштабирование не повлияло на распределение данных:

for col in scale\_cols:

col\_scaled = col + '\_scaled'

fig, ax = plt.subplots(1, 2, figsize=(8,3))

ax[0].hist(data\_all[col], 50)

ax[1].hist(data\_all[col\_scaled], 50)

ax[0].title.set\_text(col)

ax[1].title.set\_text(col\_scaled)

plt.show()

Результат работы команды представлен на рисунке 15.

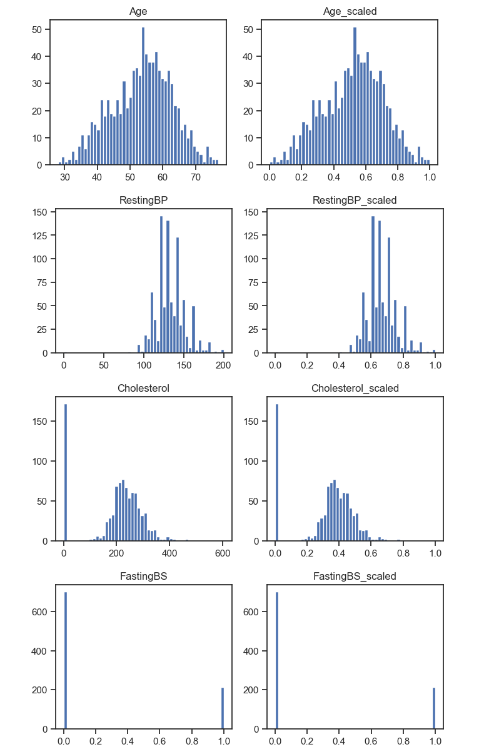
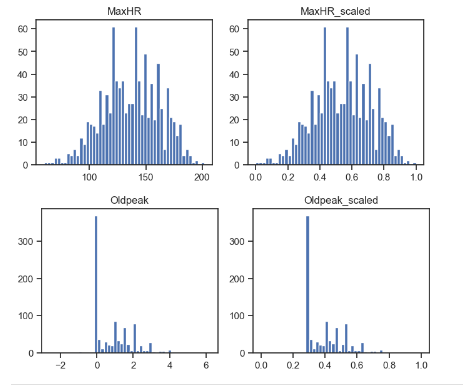


Рисунок 15— Результат выполнения команды



# **4. Корреляционный анализ данных и формирование выводов о построении моделей.**

Воспользуемся наличием тестовых выборок, включив их в корреляционную матрицу:

corr\_cols\_1 = scale\_cols + ['HeartDisease']

corr\_cols\_1

Результат работы команды представлен на рисунке 16.

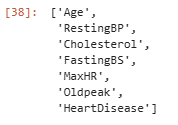


Рисунок 16 — Результат выполнения команды

Создадим новый список элементов:

scale\_cols\_postfix = [x+'\_scaled' for x in scale\_cols]

corr\_cols\_2 = scale\_cols\_postfix + ['HeartDisease']

corr\_cols\_2

Результат работы команды представлен на рисунке 17.

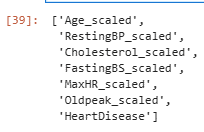


Рисунок 17 — Результат выполнения команды

Исходные данные (до масштабирования):

fig, ax = plt.subplots(figsize=(10,5))

sns.heatmap(data\_all[corr\_cols\_1].corr(), annot=True, fmt='.2f')

ax.set\_title('Исходные данные (до масштабирования)')

plt.show()

Результат работы команды представлен на рисунке 18.



Рисунок 18— Результат выполнения команды

Масштабированные данные:

fig, ax = plt.subplots(figsize=(10,5))

sns.heatmap(data\_all[corr\_cols\_2].corr(), annot=True, fmt='.2f')

ax.set\_title('Масштабированные данные')

plt.show()

Результат работы команды представлен на рисунке 19.

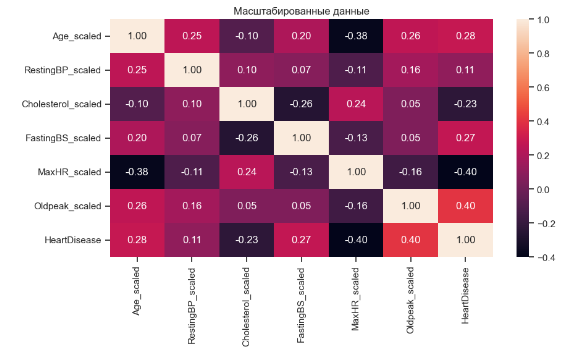


Рисунок 19— Результат выполнения команды

# **5. Выбор метрик для оценки качества моделей.**

Выбор метрик для последующей оценки качества моделей. В качестве метрик для решения задачи классификации будем использовать: Метрики, формируемые на основе матрицы ошибок:

Метрика precision: Можно переводить как точность.

precision=TPTP+FP Доля верно предсказанных классификатором положительных объектов, из всех объектов, которые классификатор верно или неверно определил как положительные.

Используется функция precision\_score.

Метрика recall (полнота): recall=TPTP+FN Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Используется функция recall\_score.

Метрика F1 -мера Для того, чтобы объединить precision и recall в единую метрику используется Fβ -мера, которая вычисляется как среднее гармоническое от precision и recall:

Fβ=(1+β2)⋅precision⋅recallprecision+recall где β определяет вес точности в метрике.

На практике чаще всего используют вариант F1-меры (которую часто называют F-мерой) при β=1 :

F1=2⋅precision⋅recallprecision+recall Для вычисления используется функция f1\_score.

Метрика ROC AUC Основана на вычислении следующих характеристик:

TPR=TPTP+FN

* True Positive Rate, откладывается по оси ординат. Совпадает с recall.

FPR=FPFP+TN

* False Positive Rate, откладывается по оси абсцисс. Показывает какую долю из объектов отрицательного класса алгоритм предсказал неверно.

Идеальная ROC-кривая проходит через точки (0,0)-(0,1)-(1,1), то есть через верхний левый угол графика.

Чем сильнее отклоняется кривая от верхнего левого угла графика, тем хуже качество классификации.

В качестве количественной метрики используется площадь под кривой - ROC AUC (Area Under the Receiver Operating Characteristic Curve). Чем ниже проходит кривая, тем меньше ее площадь и тем хуже качество классификатора.

Для получения ROC AUC используется функция roc\_auc\_score.

Разработаем класс, который позволит сохранять метрики качества построенных моделей и реализует визуализацию метрик качества:

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):

self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index, inplace = True)

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):

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)):

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()

# **6. Выбор моделей для классификации.**

Для задачи классификации будем использовать следующие модели:

* Логистическая регрессия
* Метод ближайших соседей
* Машина опорных векторов
* Решающее дерево
* Случайный лес
* Градиентный бустинг

# **7. Формирование обучающей и тестовой выборок.**

На основе масштабированных данных выделим обучающую выборку с помощью фильтра:

train\_data\_all = data\_all[data\_all['dataset']=='TRAIN']

train\_data\_all.shape

Результат работы команды представлен на рисунке 20.

Признаки и выборки для задачи классификации:



Рисунок 20 — Результат выполнения команды

task\_clas\_cols = ['Age\_scaled',

'RestingBP\_scaled',

'Cholesterol\_scaled',

'FastingBS\_scaled',

'MaxHR\_scaled',

'Oldpeak\_scaled']

clas\_X\_train = train\_data\_all[task\_clas\_cols]

clas\_Y\_train = train\_data\_all['HeartDisease']

clas\_X\_train.shape, clas\_Y\_train.shape

Результат работы команды представлен на рисунке 21.



Рисунок 21 — Результат выполнения команды

# **8. Построение базового решения (baseline) для моделей без подбора гиперпараметров.**

Модели:

clas\_models = {'LogR': LogisticRegression(),

'KNN\_5':KNeighborsClassifier(n\_neighbors=5),

'SVC':SVC(probability=True),

'Tree':DecisionTreeClassifier(),

'RF':RandomForestClassifier(),

'GB':GradientBoostingClassifier()}

Сохранение метрик:

clasMetricLogger = MetricLogger()

Отрисовка ROC-кривой

def draw\_roc\_curve(y\_true, y\_score, ax, 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)

lw = 2

ax.plot(fpr, tpr, color='darkorange',

lw=lw, label='ROC curve (area = %0.2f)' % roc\_auc\_value)

ax.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

ax.set\_xlim([0.0, 1.0])

ax.set\_xlim([0.0, 1.05])

ax.set\_xlabel('False Positive Rate')

ax.set\_ylabel('True Positive Rate')

ax.set\_title('Receiver operating characteristic')

ax.legend(loc="lower right")

Создадим класс:

from sklearn.metrics import ConfusionMatrixDisplay

def clas\_train\_model(model\_name, model, clas\_X\_train, clas\_Y\_train, clasMetricLogger):

model.fit(clas\_X\_train, clas\_Y\_train)

Y\_pred = model.predict(clas\_X\_train)

Y\_pred\_proba\_temp = model.predict\_proba(clas\_X\_train)

Y\_pred\_proba = Y\_pred\_proba\_temp[:,1]

precision = precision\_score(clas\_Y\_train, Y\_pred)

recall = recall\_score(clas\_Y\_train, Y\_pred)

f1 = f1\_score(clas\_Y\_train, Y\_pred)

roc\_auc = roc\_auc\_score(clas\_Y\_train, Y\_pred\_proba)

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)

fig, ax = plt.subplots(ncols=2, figsize=(10,5))

draw\_roc\_curve(clas\_Y\_train, Y\_pred\_proba, ax[0])

cm = confusion\_matrix(clas\_Y\_train, Y\_pred)

disp = ConfusionMatrixDisplay.from\_estimator(model, clas\_X\_train, clas\_Y\_train, display\_labels=['0','1'], cmap=plt.cm.Blues, normalize='true', ax=ax[1])

fig.suptitle(model\_name)

plt.show()

for model\_name, model in clas\_models.items():

clas\_train\_model(model\_name, model, clas\_X\_train, clas\_Y\_train, clasMetricLogger)

Результат работы команды представлен на рисунке 22.

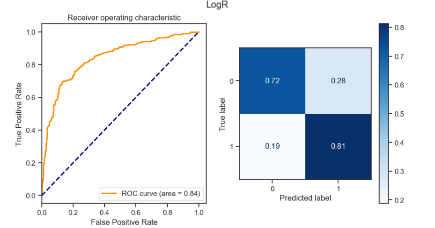
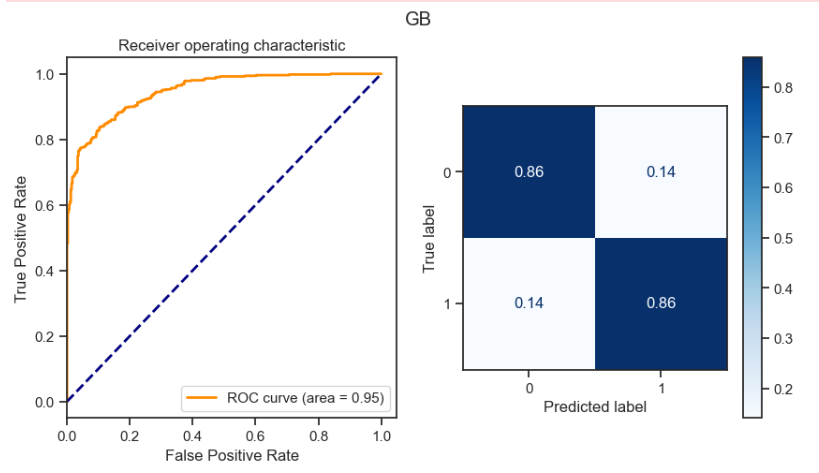
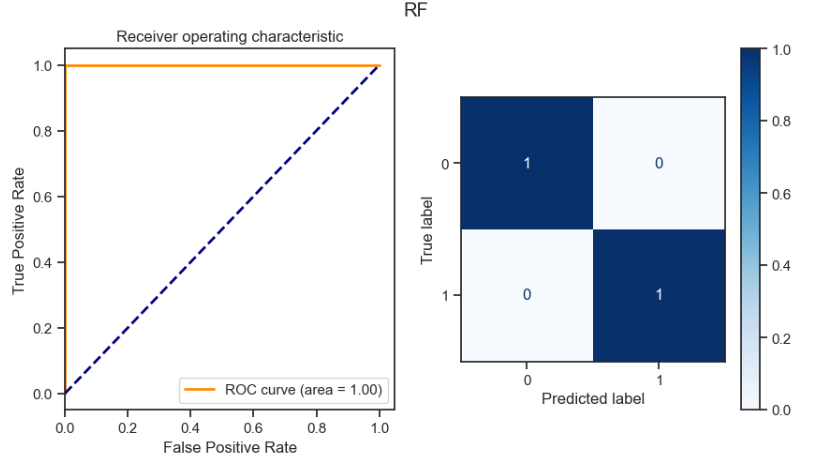
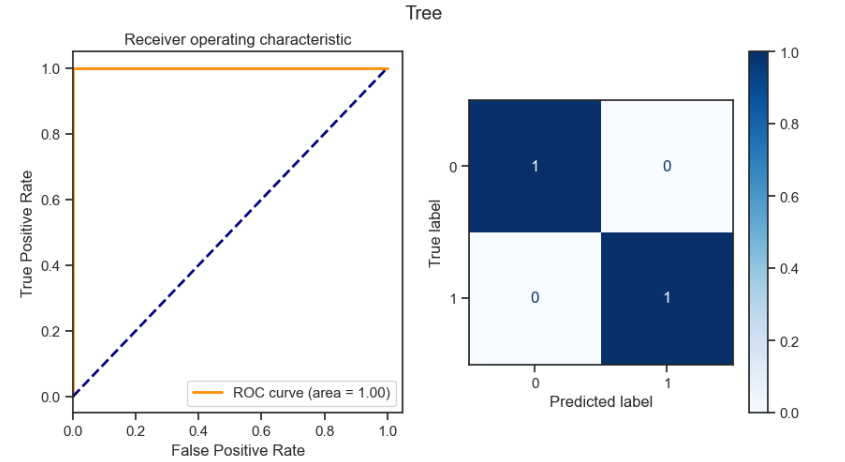
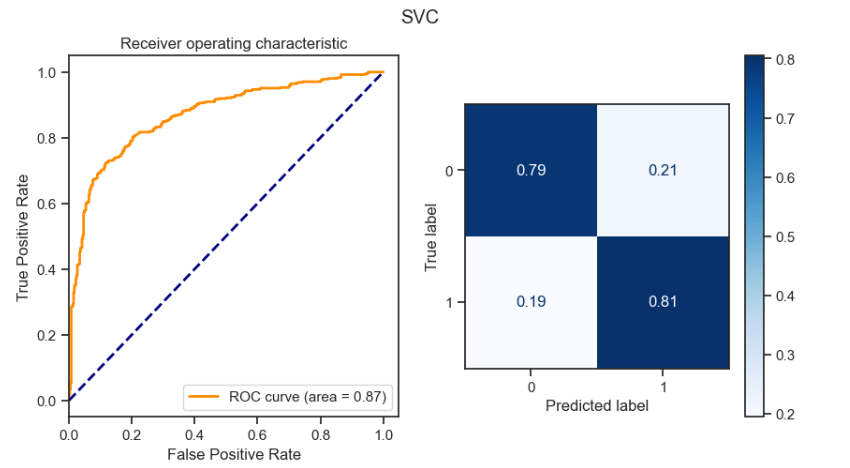
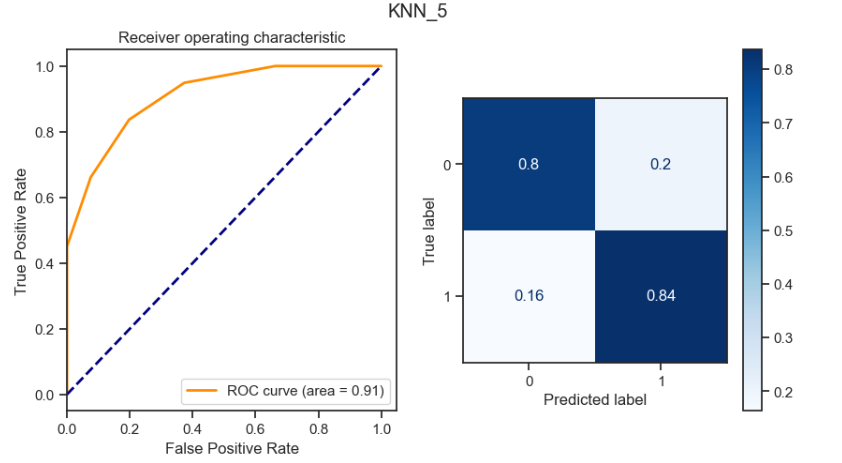


Рисунок 22— Результат выполнения команды



# **9. Подбор гиперпараметров с использованием методов кросс-валидации.**

Сгенерируем последовательность чисел:

n\_range\_list = list(range(0,750,50))

n\_range\_list[0] = 1

n\_range = np.array(n\_range\_list)

tuned\_parameters = [{'n\_neighbors': n\_range}]

tuned\_parameters

Результат работы команды представлен на рисунке 23.

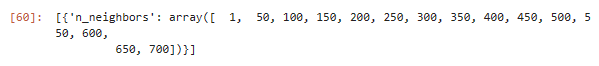


Рисунок 23 — Результат выполнения команды

Создадим экземпляр GridSearchCV, который выполняет поиск по сетке параметров (Grid Search) для модели K ближайших соседей (KNeighborsClassifier):

%%time

clf\_gs = GridSearchCV(KNeighborsClassifier(), tuned\_parameters, cv=5, scoring='roc\_auc')

clf\_gs.fit(clas\_X\_train, clas\_Y\_train)

Результат работы команды представлен на рисунке 24.

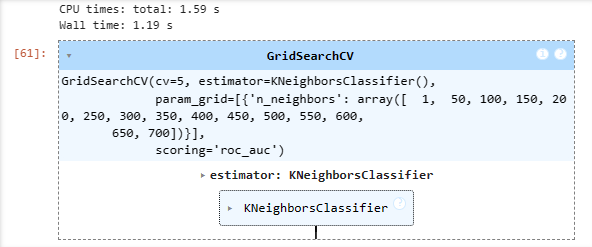


Рисунок 24 — Результат выполнения команды

Лучшая модель:

clf\_gs.best\_estimator\_

Результат работы команды представлен на рисунке 25.

Лучшее значение параметров

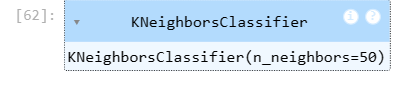


Рисунок 25 — Результат выполнения команды

clf\_gs.best\_params\_

Результат работы команды представлен на рисунке 26.



Рисунок 26 — Результат выполнения команды

Изменение качества на тестовой выборке в зависимости от К-соседей:

plt.plot(n\_range, clf\_gs.cv\_results\_['mean\_test\_score'])

Результат работы команды представлен на рисунке 27.

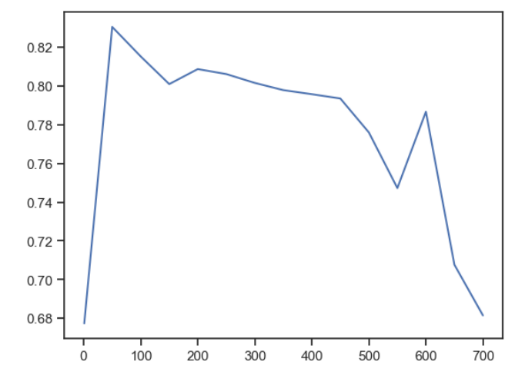


Рисунок 27— Результат выполнения команды

# **10. Проверка качества моделей с оптимальными гиперпараметрами.**

Решение задачи классификации:

clas\_models\_grid = {'KNN\_5':KNeighborsClassifier(n\_neighbors=5),

str('KNN\_' + clf\_gs\_best\_params\_txt):clf\_gs.best\_estimator\_}

for model\_name, model in clas\_models\_grid.items():

clas\_train\_model(model\_name, model, clas\_X\_train, clas\_Y\_train, clasMetricLogger)

Результат работы команды представлен на рисунке 28.

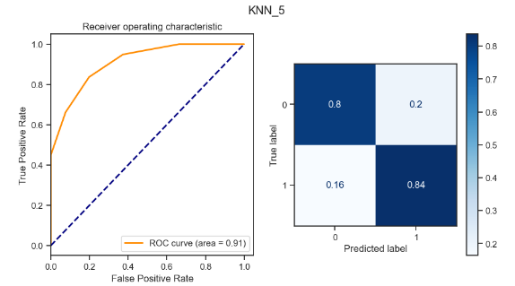
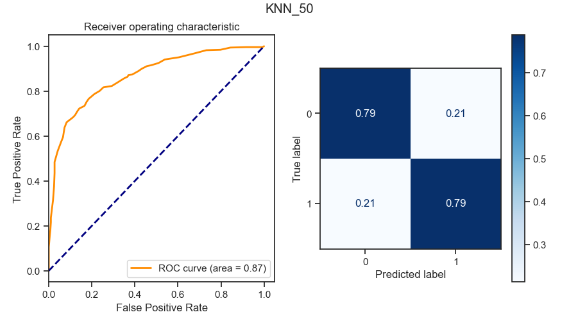


Рисунок 28— Результат выполнения команды



# **11. Формирование выводов о качестве моделей и представление результатов в виде графиков и текстовых описаний.**

Метрики качества модели:

clas\_metrics = clasMetricLogger.df['metric'].unique()

clas\_metrics

Результат работы команды представлен на рисунке 29.



Рисунок 29 — Результат выполнения команды

Построим графики метрик качества модели:

for metric in clas\_metrics:

clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))

Результат работы команды представлен на рисунке 30.

**Вывод:** на основании четырех метрик из четырех используемых, лучшими оказались модели случайного леса и дерево.

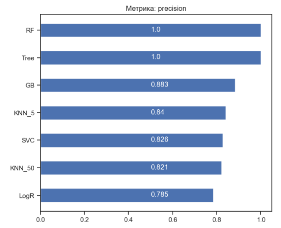
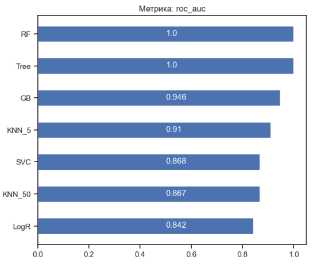
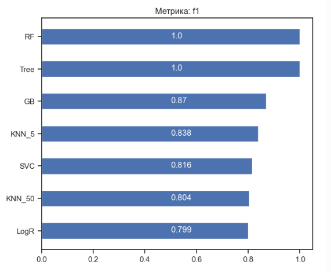
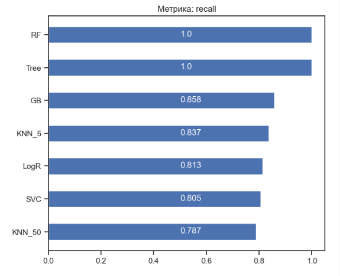


Рисунок 30— Результат выполнения команды



# **12. Создание веб-приложения для демонстрации модели машинного обучения с возможностью изменения гиперпараметров**

В рамках реализации веб-приложения используем модель RandomForest. Для отображения веб-интерфейса используем Voila:

pip install voila

from sklearn.ensemble import RandomForestRegressor

from ipywidgets import interact, fixed

from sklearn.metrics import roc\_curve, roc\_auc\_score, confusion\_matrix, precision\_score, recall\_score, f1\_score, ConfusionMatrixDisplay

clas\_X\_train = train\_data\_all[task\_clas\_cols]

clas\_Y\_train = train\_data\_all['HeartDisease']

clas\_models = {'RandomForest': RandomForestClassifier()}

clasMetricLogger = MetricLogger()

def draw\_roc\_curve(y\_true, y\_score, ax, 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)

lw = 2

ax.plot(fpr, tpr, color='darkorange', lw=lw, label='ROC curve (area = %0.2f)' % roc\_auc\_value)

ax.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')

ax.set\_xlim([0.0, 1.0])

ax.set\_ylim([0.0, 1.05])

ax.set\_xlabel('False Positive Rate')

ax.set\_ylabel('True Positive Rate')

ax.set\_title('Receiver Operating Characteristic')

ax.legend(loc="lower right")

def clas\_train\_model\_interactive(model\_name, model, clas\_X\_train, clas\_Y\_train, clasMetricLogger, n\_neighbors=50, max\_depth=None):

if max\_depth is not None:

model.set\_params(max\_depth=max\_depth)

model.set\_params(n\_estimators=n\_neighbors)

model.fit(clas\_X\_train, clas\_Y\_train)

Y\_pred = model.predict(clas\_X\_train)

Y\_pred\_proba\_temp = model.predict\_proba(clas\_X\_train)

Y\_pred\_proba = Y\_pred\_proba\_temp[:, 1]

fig, ax = plt.subplots(ncols=2, figsize=(10, 5))

draw\_roc\_curve(clas\_Y\_train, Y\_pred\_proba, ax[0])

cm = confusion\_matrix(clas\_Y\_train, Y\_pred)

disp = ConfusionMatrixDisplay.from\_estimator(model, clas\_X\_train, clas\_Y\_train, display\_labels=['0', '1'], cmap=plt.cm.Blues, normalize='true', ax=ax[1])

fig.suptitle(f'{model\_name} (n\_neighbors: {n\_neighbors}, max\_depth: {max\_depth})')

plt.show()

interact(clas\_train\_model\_interactive,

model\_name=clas\_models.keys(),

model=clas\_models.values(),

clas\_X\_train=fixed(clas\_X\_train),

clas\_Y\_train=fixed(clas\_Y\_train),

clasMetricLogger=fixed(clasMetricLogger),

n\_neighbors=(1, 100, 1),

max\_depth=(1, 20, 1))

Воспользуемся Render with Voila для отображения веб-интерфейса (Рисунок 31).

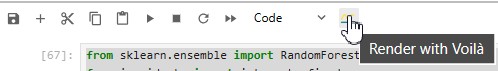


Рисунок 31 — Render with Voila

Результатом будет интерактивный виджет. При изменении гиперпараметра меняются и отображения кривых и метрик. Результат представлен на рисунке 32.

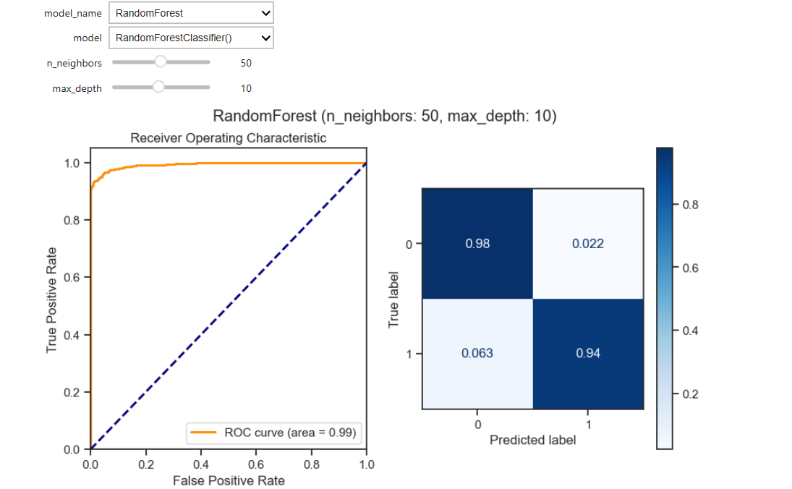


Рисунок 32— Результат выполнения команды