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 ИУ_____

КАФЕДРА ИУ5_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОМУ ПРОЕКТУ

НА ТЕМУ:

_____Решение задачи машинного обучения_____

Студент группы ИУ5-61Б___ _____ _____Мартынова Д.П.
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ _____Гапанюк Ю.Е.
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2020 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 _____
(Индекс)
_____ _____
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение курсового проекта

по дисциплине «Технологии машинного обучения»_____

Студент группы ИУ5-61Б_____

_____Мартынова Дарья Петровна_____
(Фамилия, имя, отчество)

Тема курсового проекта "Бинарная классификация"_____
_____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____
Источник тематики (кафедра, предприятие, НИР) _____

График выполнения проекта: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Задание решение задачи машинного обучения на основе материалов дисциплины. Выполняется студентом единолично.
_____
_____
_____
_____

Оформление курсового проекта:

Расчетно-пояснительная записка на __32___ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « 12 » февраля 2020 г.

Руководитель курсового проекта _____ _____ Гапанюк Ю.Е
(Подпись, дата) (И.О.Фамилия)

Студент _____ _____Мартынова Д.П.
(Подпись, дата) (И.О.Фамилия)

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

Оглавление

1. Задание

В данной курсовой работе необходимо предпринять следующие шаги:

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 трех метрик и обосновать выбор.
6. Выбор наиболее подходящих моделей для решения задачи классификации или регрессии. Необходимо использовать не менее пяти моделей, две из которых должны быть ансамблевыми.
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 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. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания. Рекомендуется постройение графиков обучения и валидации, влияния значений гиперпарметров на качество моделей и т.д.

2. Введение

Курсовой проект – самостоятельная часть учебной дисциплины «Технологии машинного обучения» – учебная и практическая исследовательская студенческая работа, направленная на решение комплексной задачи машинного обучения. Результатом курсового проекта является отчет, содержащий описания моделей, тексты программ и результаты экспериментов.

Курсовой проект опирается на знания, умения и владения, полученные студентом в рамках лекций и лабораторных работ по дисциплине.

В рамках данной курсовой работы необходимо применить навыки, полученные в течение курса «Технологии машинного обучения», и обосновать полученные результаты.

3. Постановка задачи

В данной курсовой работе ставится задача определения принадлежности звезды к классу пульсаров по различным параметрам с помощью методов машинного обучения: "Logistic Regression", "Support vector machine", "Decision tree", "Gradient boosting", "Random forest". С помощью различных метрик выбор метода, который наиболее эффективно и качественно определяет значение целевого признака.

4. Решение поставленной задачи

4.1. 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

- Описание выбранного датасета

Данный датасет состоит из данных, основанных на оцифрованных изображениях тонкоигольного аспирата грудной массы. Они описывают характеристики ядер клеток, присутствующих на изображении. С помощью этих данных можно делать выводы относительно образований в грудной клетке.

- Информация об атрибутах:
  - ✓ ID – номер записи
  - ✓ Diagnosis– диагноз: m – злокачественный, b - доброкачественный
  - ✓ radius – среднее значение расстояний от центра до точек на периметре
  - ✓ texture - стандартное отклонение значений серой шкалы
  - ✓ perimeter - периметр
  - ✓ area – площадь
  - ✓ smoothness – локальное изменение длины радиуса
  - ✓ compactness – периметр^2 / Площадь-1,0
  - ✓ concavity - выраженность вогнутых участков контура
  - ✓ concave points - количество вогнутых участков контура
  - ✓ symmetry – симметрия
  - ✓ fractal dimension - фрактальная размерность

В рассматриваемом примере будем решать ***задачу классификации***. Для этого в качестве целевого признака будем использовать атрибут " Diagnosis " (Диагноз). Поскольку признак

содержит только два значения: злокачественный или доброкачественный, то это задача бинарной классификации.

- Импорт библиотек

```
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.metrics import plot_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
#from gmdhpy import gmdh
%matplotlib inline
sns.set(style="ticks")
```

- Загрузка данных.

```
#Загружаем данные и выводим первые строки
data=pd.read_csv("cancer.csv")
data.head()
```

4.2. *Проведение разведочного анализа данных. Построение графиков, необходимых для понимания структуры данных. Анализ и заполнение пропусков в данных.*

- Основные характеристики датасета

```
#Загружаем данные и выводим первые строки
data=pd.read_csv("cancer.csv")
data.head()
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	te
0	842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	...	
1	842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	...	
2	84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	...	
3	84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	...	
4	84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	...	

5 rows × 33 columns

В данном датасете присутствует средние значение образований, их наибольшие и наименьшие значения. В нашей работе будем анализировать только средние значения, поэтому оставим в dataframe только их, удалив остальные колонки.

```
data=data[['diagnosis','radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean', 'compactness_mean', 'conc
# типы колонок
data.dtypes
```

```
diagnosis                 object
radius_mean              float64
texture_mean             float64
perimeter_mean           float64
area_mean                float64
smoothness_mean          float64
compactness_mean         float64
concavity_mean           float64
concave points_mean      float64
symmetry_mean            float64
fractal_dimension_mean   float64
dtype: object
```

```
# размер набора данных
data.shape
```

```
(569, 11)
```

```
# проверим есть ли пропущенные значения
data.isnull().sum()
```

```
diagnosis                0
radius_mean              0
texture_mean             0
perimeter_mean           0
area_mean                0
smoothness_mean          0
compactness_mean         0
concavity_mean           0
concave points_mean      0
symmetry_mean            0
fractal_dimension_mean   0
dtype: int64
```

```
data.describe()
```

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean	fract
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	

Можно увидеть, что пропуски в данных отсутствуют.

4.3. Выбор признаков, подходящих для построения моделей. Кодирование категориальных признаков. Масштабирование данных. Формирование вспомогательных признаков, улучшающих качество моделей.

Т.к. наш целевой признак на данный момент является категорийным, то его необходимо закодировать. Сделаем это с помощью LableEncoder.

```
from sklearn.preprocessing import LabelEncoder
```

```
enc = pd.DataFrame(data['diagnosis'])
enc['diagnosis'].unique()
```

```
array(['M', 'B'], dtype=object)
```

```
le = LabelEncoder()
enc_le = le.fit_transform(enc['diagnosis'])
np.unique(enc_le)
```

```
array([0, 1])
```

```
gen_enc = pd.DataFrame({'diagnosis':enc_le})
for index, row in gen_enc.iterrows():
    data.loc[index, 'diagnosis'] = row['diagnosis']
data
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mea
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.24
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.18
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.20
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.25
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.18
...	...	...	...	...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.17
565	1	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.17
566	1	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.15
567	1	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.23
568	0	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.15

569 rows × 11 columns

Еще раз посмотрим на типы столбцов датасета.

```
data['diagnosis']=data['diagnosis'].astype(str).astype(np.int64)
data.dtypes
```

```
diagnosis                 int64
radius_mean             float64
texture_mean            float64
perimeter_mean          float64
area_mean               float64
smoothness_mean         float64
compactness_mean        float64
concavity_mean          float64
concave points_mean     float64
symmetry_mean           float64
fractal_dimension_mean  float64
dtype: object
```

Категориальных признаков в датасете нет, их кодирования не требуется.

Вспомогательные признаки для улучшения качества моделей строить не будем.

Построим графики для понимания структуры данных:

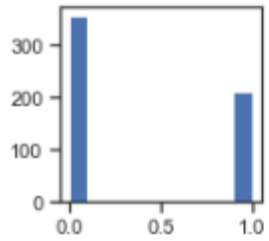
```
# Парные диаграммы
sns.pairplot(data, hue='diagnosis')
```

```
<seaborn.axisgrid.PairGrid at 0x12fc0848>
```

```
# Убедимся, что целевой признак
# для задачи бинарной классификации содержит только 0 и 1
data['diagnosis'].unique()
```

```
array([1, 0], dtype=int64)
```

```
# Оценим дисбаланс классов для целевого признака
fig, ax = plt.subplots(figsize=(2,2))
plt.hist(data['diagnosis'])
plt.show()
```




```
# посчитаем дисбаланс классов
total = data.shape[0]
class_0, class_1 = data['diagnosis'].value_counts()
print('Класс 0 составляет {}%, а класс 1 составляет {}%.'
      .format(round(class_0 / total, 4)*100, round(class_1 / total, 4)*100))
```

```
Класс 0 составляет 62.739999999999995%, а класс 1 составляет 37.26%.
```

Можно заметить, что дисбаланс классов в данном случае присутствует, но является несущественным.

Выполним масштабирование данных.

```
data.head()
```

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	symmetry_mean
0	1	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419
1	1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812
2	1	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069
3	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597
4	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809

5 rows × 21 columns

```
# Числовые колонки для масштабирования
scale_cols = ['radius_mean', 'texture_mean', 'perimeter_mean', 'area_mean', 'smoothness_mean','compactness_mean',
              'concavity_mean', 'concave points_mean', 'symmetry_mean', 'fractal_dimension_mean']
```

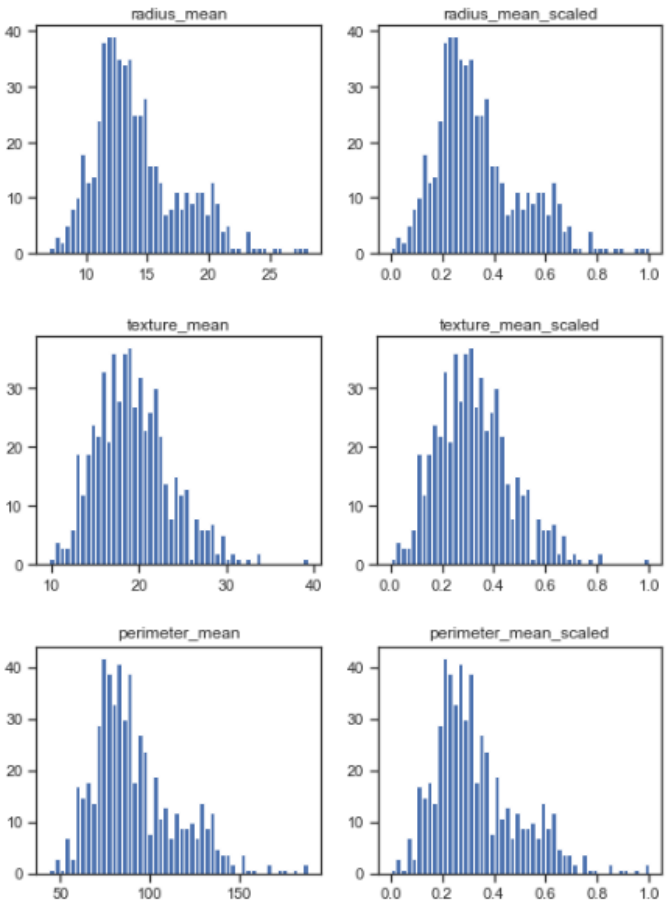
```
from sklearn.preprocessing import MinMaxScaler
sc1 = MinMaxScaler()
sc1_data = sc1.fit_transform(data[scale_cols])
```

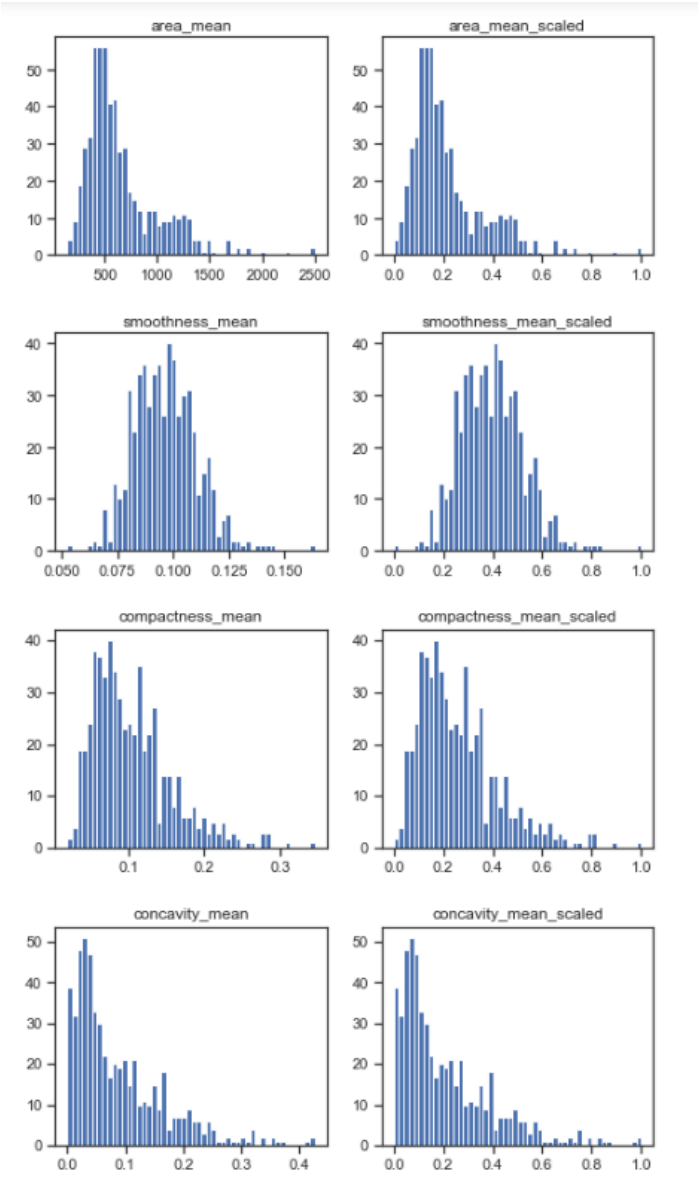
```
# Добавим масштабированные данные в набор данных
for i in range(len(scale_cols)):
    col = scale_cols[i]
    new_col_name = col + '_scaled'
    data[new_col_name] = sc1_data[:,i]
```

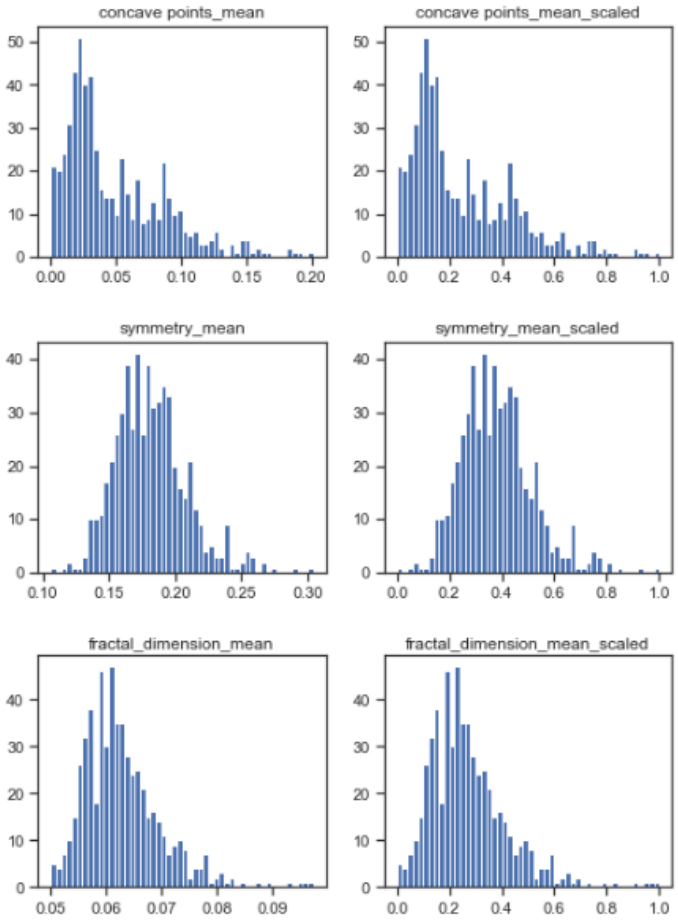
```
data.head()
```

radius_mean_scaled	texture_mean_scaled	perimeter_mean_scaled	area_mean_scaled	smoothness_mean_scaled	compactness_mean_scaled	concavity_mean_sca
0.521037	0.022658	0.545989	0.363733	0.593753	0.792037	0.703
0.643144	0.272574	0.615783	0.501591	0.289880	0.181768	0.203
0.601496	0.390260	0.595743	0.449417	0.514309	0.431017	0.462
0.210090	0.360839	0.233501	0.102906	0.811321	0.811361	0.565
0.629893	0.156578	0.630986	0.489290	0.430351	0.347893	0.463

```
# 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[col], 50)
    ax[1].hist(data[col_scaled], 50)
    ax[0].title.set_text(col)
    ax[1].title.set_text(col_scaled)
    plt.show()
```







4.4. Проведение корреляционного анализа данных. Формирование промежуточных выводов о возможности построения моделей машинного обучения.

```
corr_cols_1 = scale_cols + ['diagnosis']
corr_cols_1
```

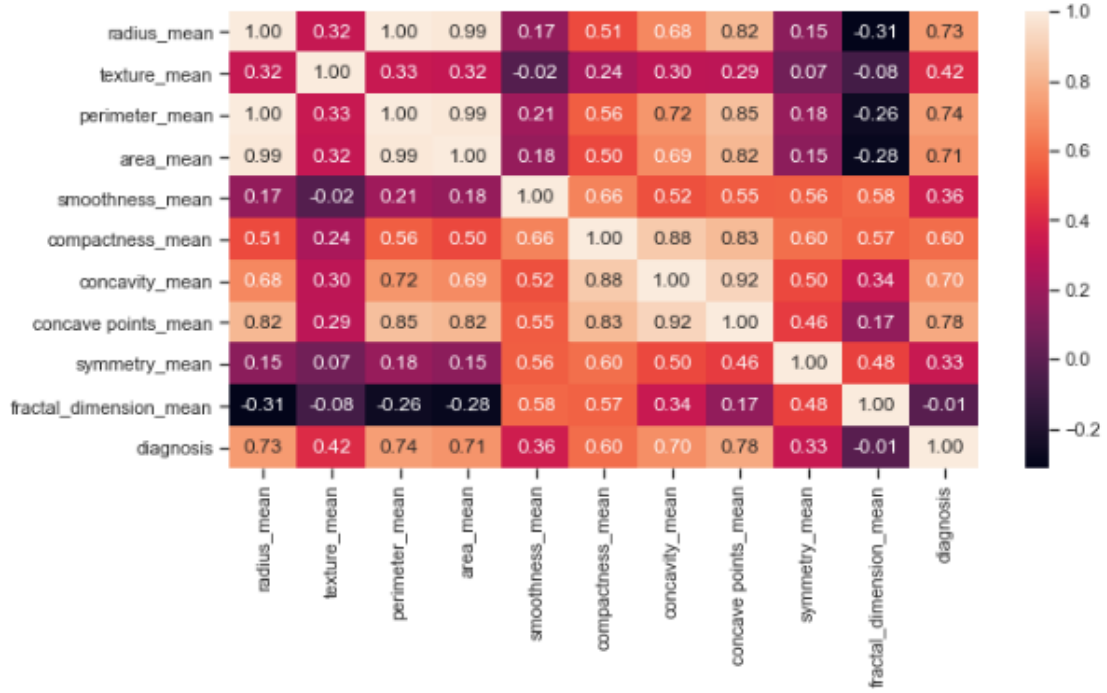
```
['radius_mean',
 'texture_mean',
 'perimeter_mean',
 'area_mean',
 'smoothness_mean',
 'compactness_mean',
 'concavity_mean',
 'concave points_mean',
 'symmetry_mean',
 'fractal_dimension_mean',
 'diagnosis']
```

```
scale_cols_postfix = [x+'_scaled' for x in scale_cols]
corr_cols_2 = scale_cols_postfix + ['diagnosis']
corr_cols_2
```

```
['radius_mean_scaled',
 'texture_mean_scaled',
 'perimeter_mean_scaled',
 'area_mean_scaled',
 'smoothness_mean_scaled',
 'compactness_mean_scaled',
 'concavity_mean_scaled',
 'concave points_mean_scaled',
 'symmetry_mean_scaled',
 'fractal_dimension_mean_scaled',
 'diagnosis']
```

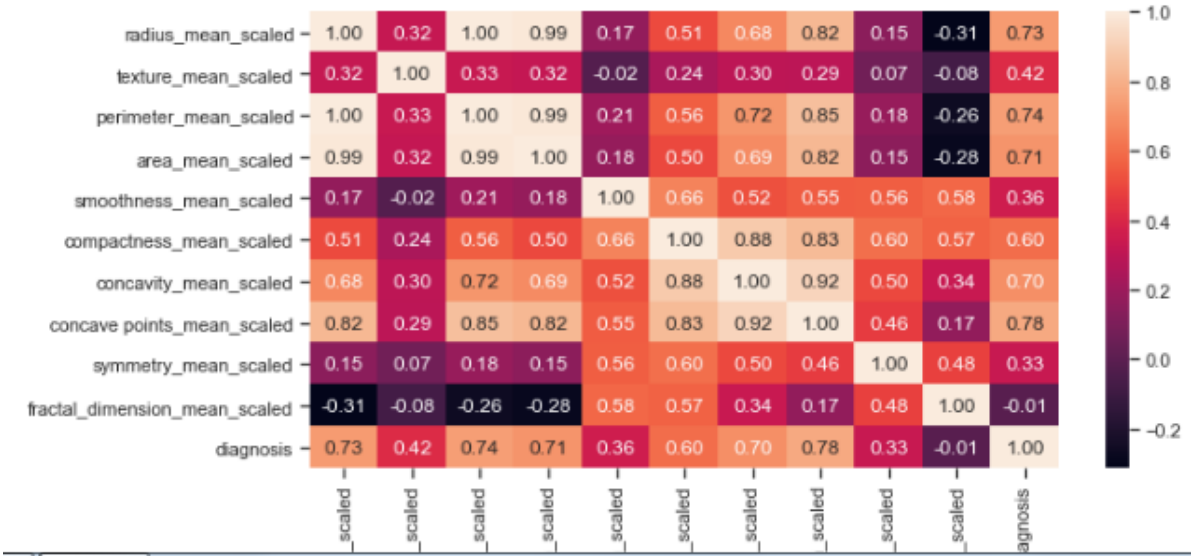
```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x140d4508>
```




```
fig, ax = plt.subplots(figsize=(10,5))
sns.heatmap(data[corr_cols_2].corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1692f508>
```



На основе корреляционной матрицы можно сделать следующие выводы:

1. Корреляционные матрицы для исходных и масштабированных данных совпадают.

2. Целевой признак классификации " diagnosis" наиболее сильно коррелирует с признаками

«radius_mean» (средний радиус) - 0.73
«perimeter_mean» (средний периметр) - 0.74
«area_mean» (средняя площадь) - 0.71
«concavity_mean» (средняя выгнутость участков контура) - 0.70
«concave points_mean» (среднее количество выгнутых участков контура) - 0.78

Эти признаки обязательно следует оставить в модели классификации.

3. Однако признаки «radius_mean» (средний радиус), «perimeter_mean» (средний периметр) и «area_mean» (средняя площадь) имеют корреляцию, близкую по модулю к 1, поэтому следует исключить «radius_mean» и «area_mean» из-за меньшей корреляции.

4. Признаки «fractal_dimension_mean» (средняя фрактальная размерность), «smoothness_mean» (среднее изменение длины радиуса) и «symmetry_mean» (средняя симметрия) слишком слабо коррелируют с целевым признаком, поэтому их следует исключить из модели, так как они могут ухудшить её качество.

5. Достаточно большие по модулю значения коэффициентов корреляции свидетельствуют о значимой корреляции между исходными признаками и целевым признаком. На основании корреляционной матрицы можно сделать вывод о том, что данные позволяют построить модель машинного обучения.

```
data = data[corr_cols_2].drop(['area_mean_scaled', 'radius_mean_scaled', 'fractal_dimension_mean_scaled',
                               'smoothness_mean_scaled', 'symmetry_mean_scaled'], axis=1)
data
```

	texture_mean_scaled	perimeter_mean_scaled	compactness_mean_scaled	concavity_mean_scaled	concave points_mean_scaled	diagnosis
0	0.022658	0.545989	0.792037	0.703140	0.731113	1
1	0.272574	0.615783	0.181768	0.203608	0.348757	1
2	0.390260	0.595743	0.431017	0.462512	0.635686	1
3	0.360839	0.233501	0.811361	0.565604	0.522863	1
4	0.156578	0.630986	0.347893	0.463918	0.518390	1
...	...	...	...	...	...	...
564	0.428813	0.678668	0.296055	0.571462	0.690358	1
565	0.626987	0.604036	0.257714	0.337395	0.486630	1
566	0.621238	0.445788	0.254340	0.216753	0.263519	1
567	0.663510	0.665538	0.790197	0.823336	0.755467	1
568	0.501522	0.028540	0.074351	0.000000	0.000000	0

569 rows × 6 columns

4.5. Выбор метрик для последующей оценки качества моделей.

В качестве метрик для решения задачи классификации будем использовать:

1. Метрика precision:

$$precision = \frac{TP}{TP+FP}$$

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

Используется функция *precision_score.*

2. Метрика recall (полнота):

$$recall = \frac{TP}{TP+FN}$$

Доля верно предсказанных классификатором положительных объектов, из всех действительно положительных объектов.

Используется функция *recall_score*.

3. Метрика F1-мера

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

$$F_\beta = (1+\beta^2) \cdot \frac{precision \cdot recall}{precision+recall}$$

где β определяет вес точности в метрике.

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

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

Для вычисления используется функция *f1_score*.

4. ***Метрика ROC AUC***

Используется для оценки качества бинарной классификации. Основана на вычислении следующих характеристик:

$$TPR = \frac{TP}{TP+FN}$$
$$FPR = \frac{FP}{FP+TN}$$

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

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

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.*

Сохранение и визуализация метрик

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):
        """
        Добавление значения
        """
        # Удаление значения если оно уже было ранее добавлено
        self.df.drop(self.df[(self.df['metric']==metric)&(self.df['alg']==alg)].index, inplace = True)
        # Добавление нового значения
        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):
        """
        Формирование данных с фильтром по метрике
        """
        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)):
        """
        Вывод графика
        """
        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()
```

4.6. *Выбор наиболее подходящих моделей для решения задачи классификации или регрессии.*

Для задачи классификации будем использовать следующие модели:

- Логистическая регрессия

Метод, используемый для решения задачи бинарной классификации.

Метод выдает вероятность принадлежности объекта к нулевому/единичному классам.

Используется класс *LogisticRegression*.

- Машина опорных векторов

Основная идея метода — перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

Для решения задачи классификации используем класс:

SVC - основной классификатор на основе SVM. Поддерживает различные ядра.

- Решающее дерево

Для текущего выбранного признака (колонки) из N признаков построить все варианты ветвления по значениям (для категориальных признаков) или по диапазонам значений (для числовых признаков).

Если подвыборке соответствует единственное значение целевого признака, то в дерево добавляется терминальный лист, который соответствует предсказанному значению.

Если в подвыборке больше одного значения целевого признака, то предыдущие пункты выполняются рекурсивно для подвыборки.

Для решения задачи классификации используется класс *DecisionTreeClassifier*.

- Случайный лес (ансамблевая)

Случайный лес можно рассматривать как алгоритмом бэггинга над решающими деревьями.

Но при этом каждое решающее дерево строится на случайно выбранном подмножестве признаков. Эта особенность называется "feature bagging" и основана на методе случайных подпространств.

Случайный лес для задача классификации реализуется в scikit-learn с помощью класса *RandomForestClassifier*.

Задание параметра n_jobs=-1 распараллеливает алгоритм на максимально возможное количество процессоров.

- Градиентный бустинг (ансамблевая)

В отличие от методов бэггинга и случайного леса, которые ориентированы прежде всего на минимизизацию дисперсии (Variance), методы бустинга ориентированы прежде всего на минимизацию смещения (Bias) и, отчасти, на минимизизацию дисперсии.

Исторически первым полноценным алгоритмом бустинга считается алгоритм AdaBoost.

AdaBoost реализуется в scikit-learn с помощью класса *AdaBoostClassifier* для задач классификации.

4.7. Формирование обучающей и тестовой выборок на основе исходного набора данных.

Возьмем наши масштабированные данные и выделим обучающую и тестовую. Т.к. изначально датасет небольшой, то для тестовой выборки оставим 5%.

```
target = data['diagnosis']
data = data.drop('diagnosis', axis = 1)
```

```
data.columns
```

```
Index(['texture_mean_scaled', 'perimeter_mean_scaled',
       'compactness_mean_scaled', 'concavity_mean_scaled',
       'concave points_mean_scaled'],
      dtype='object')
```

```
data.head()
```

	texture_mean_scaled	perimeter_mean_scaled	compactness_mean_scaled	concavity_mean_scaled	concave points_mean_scaled
0	0.022658	0.545989	0.792037	0.703140	0.731113
1	0.272574	0.615783	0.181768	0.203608	0.348757
2	0.390260	0.595743	0.431017	0.462512	0.635686
3	0.360839	0.233501	0.811361	0.565604	0.522863
4	0.156578	0.630986	0.347893	0.463918	0.518390

```
#деление на тестовую и обучающую выборку
clas_X_train, clas_X_test, clas_Y_train, clas_Y_test = train_test_split(
    data, target, test_size=0.05, random_state=1)
clas_X_train.shape, clas_X_test.shape, clas_Y_train.shape, clas_Y_test.shape
```

```
((540, 5), (29, 5), (540,), (29,))
```

4.8. *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.*

```
# Модели
clas_models = {'LogR': LogisticRegression(),
               'SVC':SVC(),
               'Tree':DecisionTreeClassifier(),
               'RF':RandomForestClassifier(),
               'GB':GradientBoostingClassifier()}
```

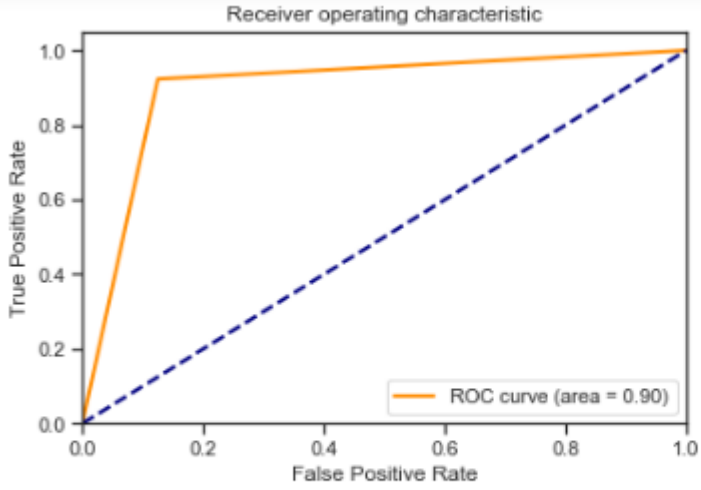
```
# Сохранение метрик
clasMetricLogger = MetricLogger()
```

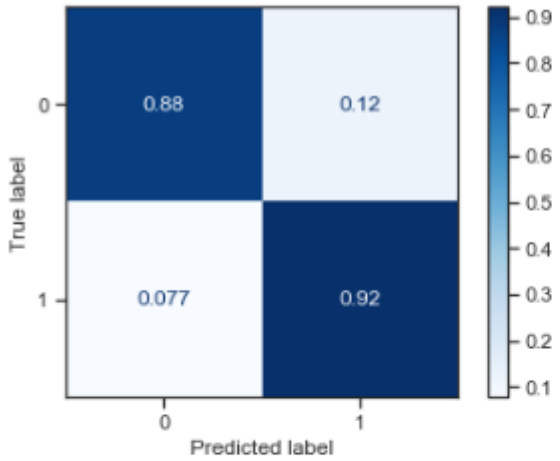
```
def clas_train_model(model_name, model, clasMetricLogger):
    model.fit(clas_X_train, clas_Y_train)
    Y_pred = model.predict(clas_X_test)
    precision = precision_score(clas_Y_test.values, Y_pred)
    recall = recall_score(clas_Y_test.values, Y_pred)
    f1 = f1_score(clas_Y_test.values, Y_pred)
    roc_auc = roc_auc_score(clas_Y_test.values, Y_pred)

    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)

    print('****************************************************')
    print(model)
    print('****************************************************')
    draw_roc_curve(clas_Y_test.values, Y_pred)

    plot_confusion_matrix(model, clas_X_test, clas_Y_test.values,
                          display_labels=['0','1'],
                          cmap=plt.cm.Blues, normalize='true')
    plt.show()
```

```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, 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)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic')
    plt.legend(loc="lower right")
    plt.show()
```

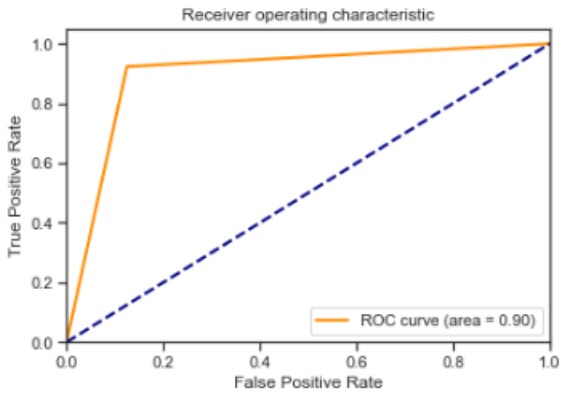
```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

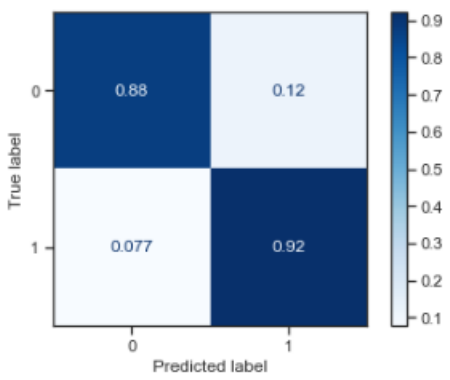
```
****************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
****************************************************
```



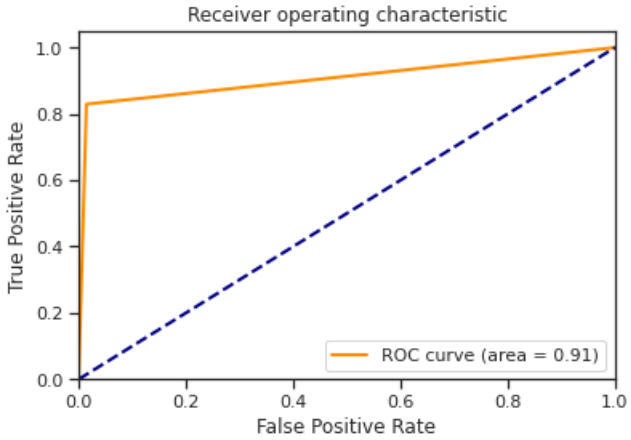


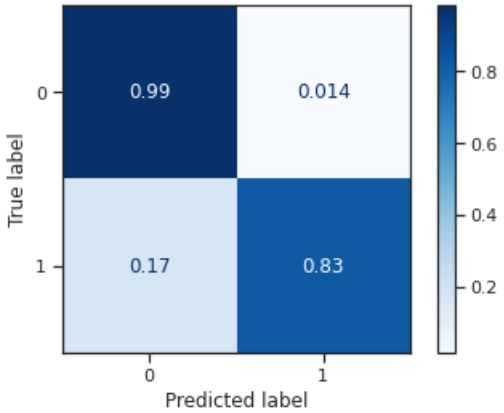

```
**********************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
**********************************************************
```



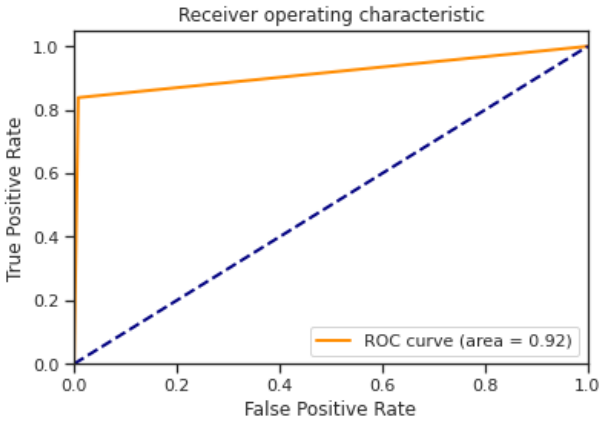


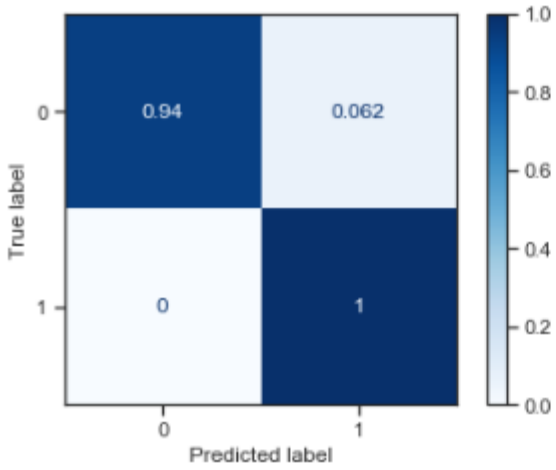
```
**********************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
**********************************************************
```



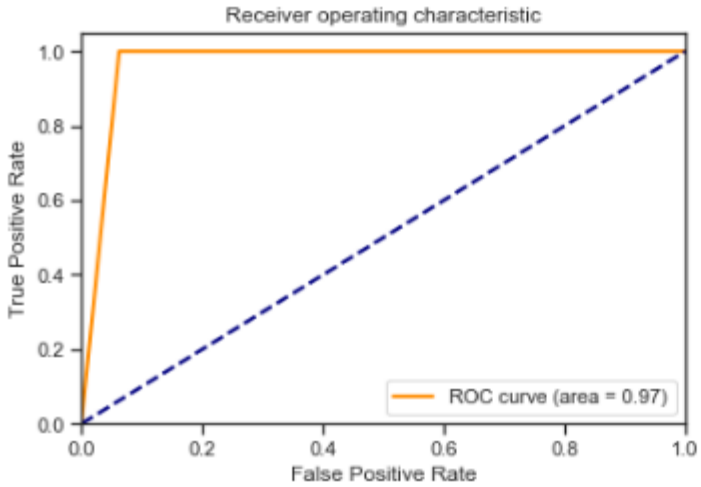


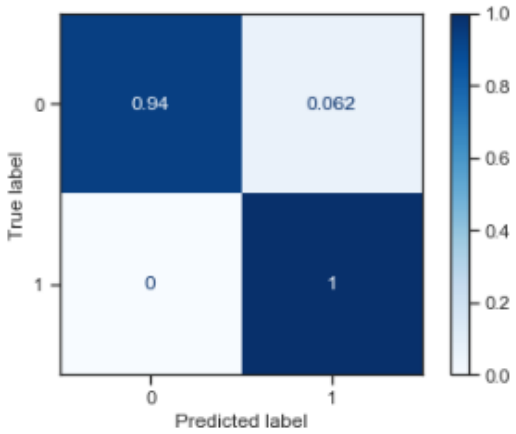

```
**********************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
**********************************************************
```






```
**********************************************************
GradientBoostingClassifier(ccp_alpha=0.0, 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=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
**********************************************************
```





4.9. *Подбор гиперпараметров для выбранных моделей. Рекомендуется использовать методы кросс-валидации. В зависимости от используемой библиотеки можно применять функцию GridSearchCV, использовать перебор параметров в цикле, или использовать другие методы.*

```
parameters = {'penalty':['l1', 'l2', 'elasticnet'],
              'C':[0.1,0.8,0.9,1,1.1,1.2,1.3,1.4]}
clf_gs_log = GridSearchCV(LogisticRegression(), parameters, cv=5, scoring='roc_auc')
clf_gs_log.fit(clas_X_train, clas_Y_train)
```

```
C:\Users\Dasik\anaconda3\lib\site-packages\sklearn\model_selection\_validation.py:536: FitFailedWarning: Estimator fit faile
d. The score on this train-test partition for these parameters will be set to nan. Details:
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got elasticnet penalty.

  FitFailedWarning)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.1, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4],
                         'penalty': ['l1', 'l2', 'elasticnet']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

```
# Лучшая модель
clf_gs_log.best_estimator_
```

```
LogisticRegression(C=1.4, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
# Лучшее значение параметров
clf_gs_log.best_params_
```

```
{'C': 1.4, 'penalty': 'l2'}
```

```
parameters = {'gamma':[200,160,150,130,120,110,100,50, 25, 10, 1],
              'kernel': ['linear', 'rbf']}
clf_gs_svm_svc = GridSearchCV(SVC(), parameters, cv=5, scoring='roc_auc')
clf_gs_svm_svc.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
             param_grid={'gamma': [200, 160, 150, 130, 120, 110, 100, 50, 25,
                                   10, 1],
                         'kernel': ['linear', 'rbf']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

```
# Лучшая модель
clf_gs_svm_svc.best_estimator_
```

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf', max_iter=-1,
    probability=False, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
```

```
# Лучшее значение параметров
clf_gs_svm_svc.best_params_
```

```
{'gamma': 1, 'kernel': 'rbf'}
```

```
parameters = {'max_depth':[20,15,10,6,5,4,3], 'min_samples_split':[10,8,6,5,4,3,2]}
clf_gs_decision_tree = GridSearchCV(DecisionTreeClassifier(), parameters, cv=5, scoring='roc_auc')
clf_gs_decision_tree.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, 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='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [20, 15, 10, 6, 5, 4, 3],
                         'min_samples_split': [10, 8, 6, 5, 4, 3, 2]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

```
clf_gs_decision_tree.best_estimator_
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=6,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

```
# Лучшее значение параметров
clf_gs_decision_tree.best_params_
```

```
{'max_depth': 4, 'min_samples_split': 6}
```

```
parameters_random_forest = {'n_estimators':[1, 3, 5, 6, 7, 8, 10, 20],
                            'max_depth':[1, 3, 5, 6, 7, 8, 10],
                            'random_state':[0, 2, 4, 6, 8, 10, 15]}
best_random_forest = GridSearchCV(RandomForestClassifier(), parameters_random_forest, cv=5, scoring='roc_auc')
best_random_forest.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=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=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [1, 3, 5, 6, 7, 8, 10],
                         'n_estimators': [1, 3, 5, 6, 7, 8, 10, 20],
                         'random_state': [0, 2, 4, 6, 8, 10, 15]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

```
# Лучшая модель
best_random_forest.best_estimator_
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=20,
                       n_jobs=None, oob_score=False, random_state=15, verbose=0,
                       warm_start=False)
```

```
best_random_forest.best_params_
```

```
{'max_depth': 5, 'n_estimators': 20, 'random_state': 15}
```

```
parameters_gradient_boosting = {'n_estimators':[2, 3, 5, 7, 10, 15, 17, 20, 23, 25, 27],
                                'max_depth':[1, 2 ,3, 5, 7, 9, 10, 15]}
best_gradient_boosting = GridSearchCV(GradientBoostingClassifier(), parameters_gradient_boosting, cv=5, scoring='roc_auc')
best_gradient_boosting.fit(clas_X_train, clas_Y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
             estimator=GradientBoostingClassifier(ccp_alpha=0.0,
                                                  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=100,
                                                  n_iter_no_change=None,
                                                  presort='deprecated',
                                                  random_state=None,
                                                  subsample=1.0, tol=0.0001,
                                                  validation_fraction=0.1,
                                                  verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [1, 2, 3, 5, 7, 9, 10, 15],
                         'n_estimators': [2, 3, 5, 7, 10, 15, 17, 20, 23, 25,
                                          27]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='roc_auc', verbose=0)
```

```
# Лучшая модель
best_gradient_boosting.best_estimator_
```

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           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=25,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

```
best_gradient_boosting.best_params_
```

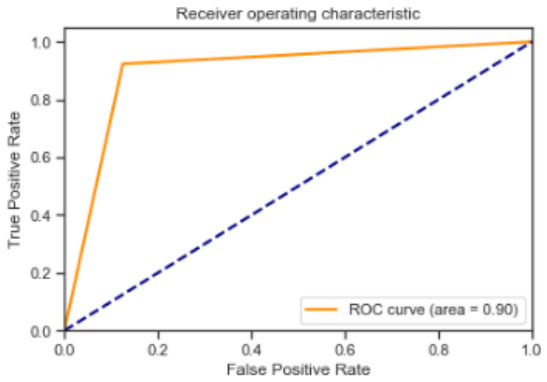
```
{'max_depth': 2, 'n_estimators': 25}
```

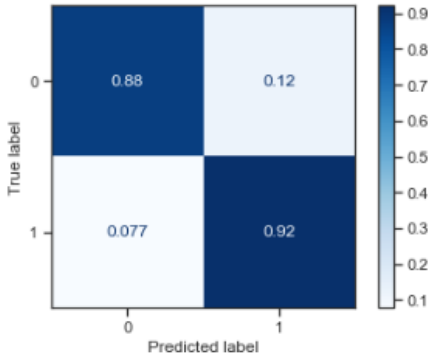
4.10. *Повторение пункта 8 для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.*

```
: # Новые модели с подобранными гиперпарамаетрами
  clas_models_grid = {'LogR':clf_gs_log.best_estimator_,
                      'SVC':clf_gs_svm_svc.best_estimator_,
                      'Tree':clf_gs_decision_tree.best_estimator_,
                      'RF':best_random_forest.best_estimator_,
                      'GB':best_gradient_boosting.best_estimator_}
```

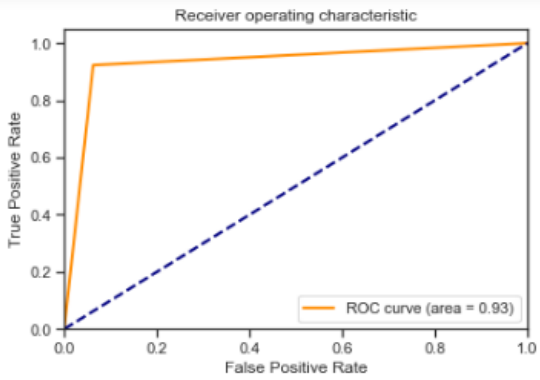
```
: for model_name, model in clas_models_grid.items():
      clas_train_model(model_name, model, clasMetricLogger)
```

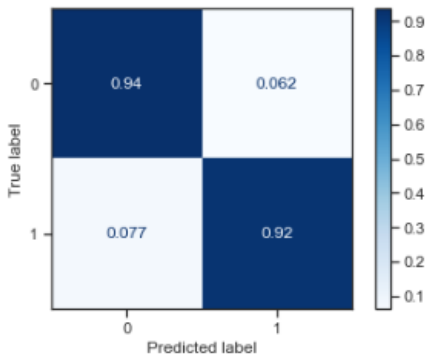
```
  ****************************************************
  LogisticRegression(C=1.4, class_weight=None, dual=False, fit_intercept=True,
                     intercept_scaling=1, l1_ratio=None, max_iter=100,
                     multi_class='auto', n_jobs=None, penalty='l2',
                     random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                     warm_start=False)
  ****************************************************
```



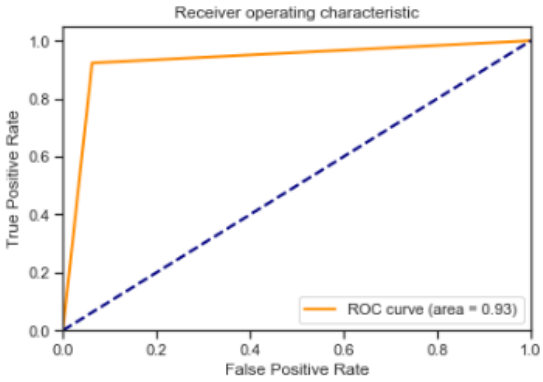


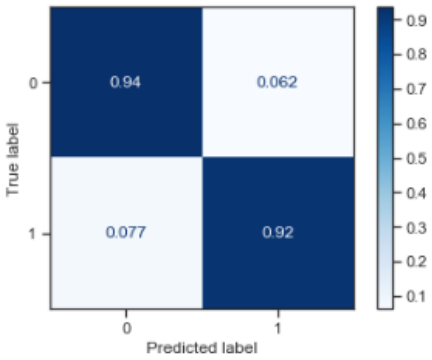

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='rbf', max_iter=-1,
    probability=False, random_state=None, shrinking=True, tol=0.001,
    verbose=False)
**************************************************
```



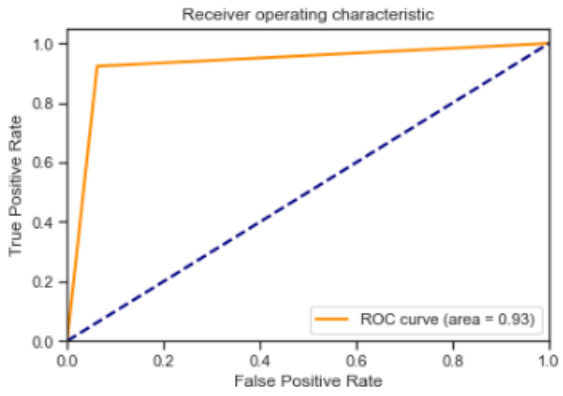


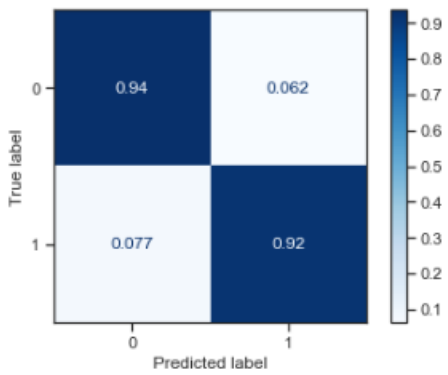

```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=6,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
**************************************************
```



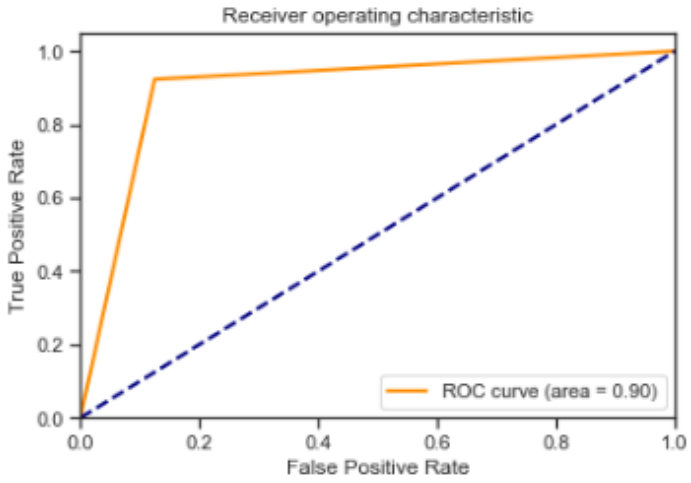


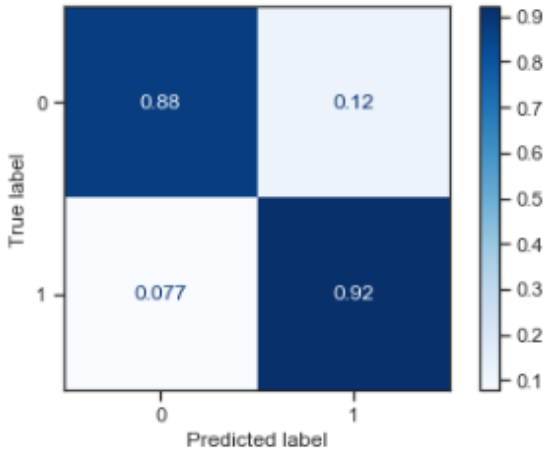

```
**********************************************************
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=5, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=20,
                       n_jobs=None, oob_score=False, random_state=15, verbose=0,
                       warm_start=False)
**********************************************************
```





```
**************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=2,
                           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=25,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
**************************************************
```



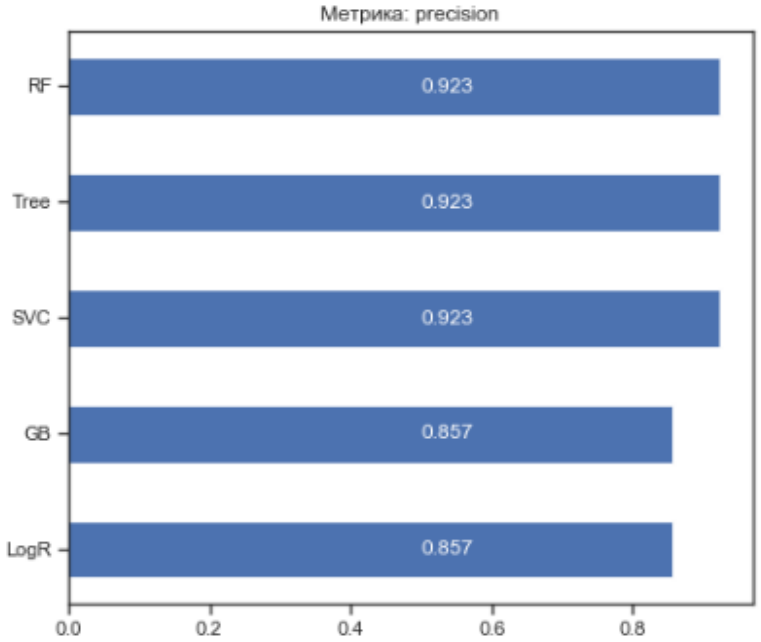


4.11. *Формирование выводов о качестве построенных моделей на основе выбранных метрик. Результаты сравнения качества рекомендуется отобразить в виде графиков и сделать выводы в форме текстового описания.*

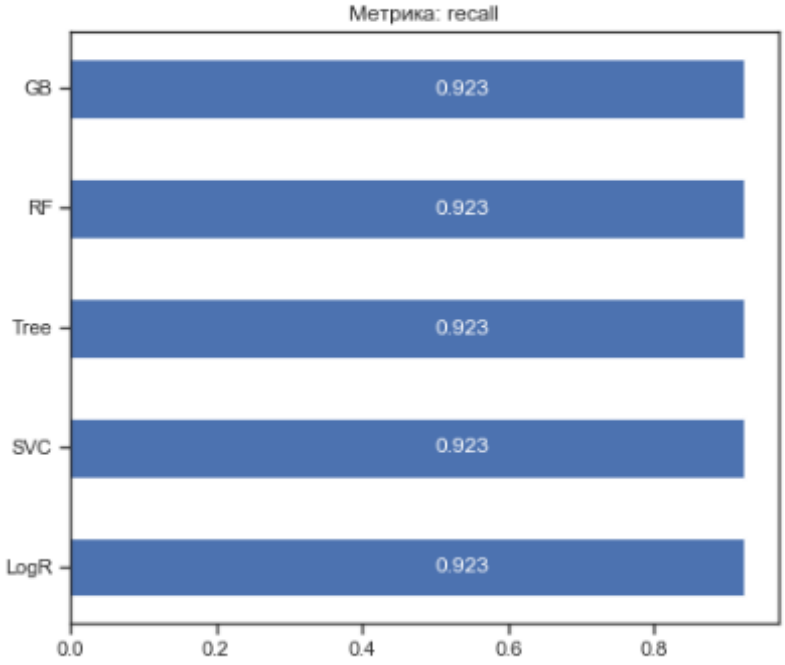
```
# Метрики качества модели
clas_metrics = clasMetricLogger.df['metric'].unique()
clas_metrics
```

```
array(['precision', 'recall', 'f1', 'roc_auc'], dtype=object)
```

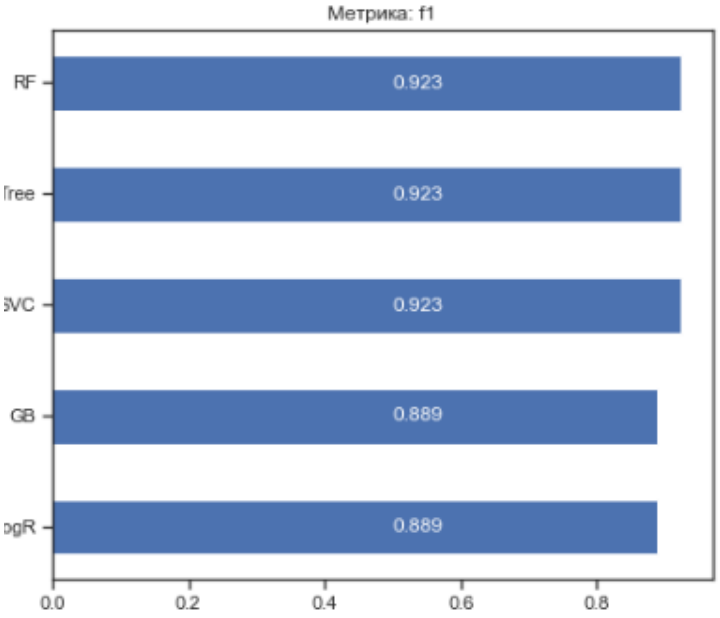
```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```



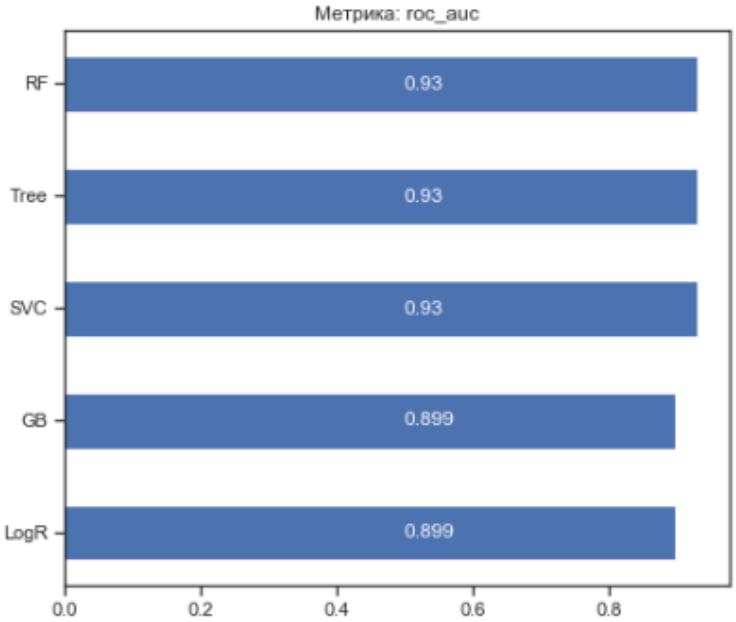
По метрике precision лучшими моделями являются: *Случайный лес*, *Решающее дерево* и *Метод опорных векторов*



По метрике recall все модели показали одинаково высокий результат.



По метрике f1 лучшими моделями являются: *Случайный лес, Решающее дерево и Метод опорных векторов*



По метрике ROC AUC лучшими моделями также являются: *Случайный лес, Решающее дерево и Метод опорных векторов*

Вывод: на основании трех метрик из четырех используемых, лучшими оказались модели *Случайного леса, Решающего дерева и Метода опорных векторов*

5. Заключение

Из всех рассмотренных алгоритмов: "Logistic Regression", "Support vector machine", "Decision tree", "Gradient boosting", "Random forest" для модели классификации звезды на принадлежность к классу пульсаров наиболее эффективным оказался алгоритм случайного леса, т.е. "Random forest". Как известно Random forest борется с переобучением модели, следовательно можно сделать вывод о том, что датасет является довольно разрозненным, поэтому другие методы могли привести к возникновению проблемы переобучения, а "Random forest" успешно обошёл эту проблему.

6. Список литературы

1. Репозиторий курса "Технологии машинного обучения", бакалавриат, 6 семестр. Лекции по теории машинного обучения. Ю.Е. Гапанюк [Электронный ресурс]. – Электрон. дан. - URL: https://github.com/ugapanyuk/ml_course_2020/wiki/COURSE_TMO (дата обращения: 03.06.2020)
2. Breast Cancer Wisconsin [Электронный ресурс]. – Электрон. дан. - URL: https://www.kaggle.com/uciml/breast-cancer-wisconsin-data (дата обращения: 31.05.2020)
3. Машинное обучение (часть 1). А.М.Миронов [Электронный ресурс]. – Электрон. дан. - URL: http://www.intsys.msu.ru/staff/mironov/machine_learning_vol1.pdf (дата обращения: 31.05.2020)
4. Scikit learn[Электронный ресурс]. – Электрон. дан. - URL: https://scikit-learn.org/stable/index.html (дата обращения: 31.05.2020)