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»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

НА ТЕМУ:

Использование методов машинного обучения для решения задачи классификации

Студент группы ИУ5-31М _____ Крюков Г.М.
(Группа) (Подпись, дата) (И.О.Фамилия)

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

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

Москва, 2022 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 научно-исследовательской работы

по теме Использование методов машинного обучения для решения задачи классификации

Студент группы ИУ5-31М

Крюков Геннадий Максимович
(Фамилия, имя, отчество)

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

Источник тематики (кафедра, предприятие, НИР) кафедра

График выполнения НИР: 25% к 4 нед., 50% к 8 нед., 75% к 12 нед., 100% к 17 нед.

Техническое задание _____

Оформление научно-исследовательской работы:

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

Дата выдачи задания « ___ » _____ 20__ г.

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

Студент _____ Г.М. Крюков
(Подпись, дата) (И.О.Фамилия)

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

Содержание

Введение

Пульсары - это редкий тип нейтронных звёзд, которые производят радиоизлучение, обнаруживаемое здесь, на Земле. Они представляют значительный научный интерес для исследования пространства-времени, межзвездной среды и состояний материи.

Когда пульсары вращаются, их луч их излучения проносится по небу, и когда он пересекает линию видимости, образуется обнаруживаемая картина широкополосного радиоизлучения. При динамичном вращении пульсаров эта картина периодически повторяется. Таким образом, поиск пульсаров предполагает поиск периодических радиосигналов с помощью больших радиотелескопов.

Каждый пульсар производит уникальную картину излучения, которая слегка меняется с каждым вращением. Таким образом, потенциальное обнаружение сигнала, рассматриваемого как "кандидат", усредняется по многим вращениям пульсара, определяемым длиной наблюдения. В отсутствие дополнительной информации каждый кандидат потенциально мог бы описать реальный пульсар. Однако на практике почти все обнаружения вызваны радиочастотными помехами (RFI) и шумом, что затрудняет поиск законных сигналов.

В данной работе ставится задача определения принадлежности звезды к классу пульсаров по различным параметрам с помощью методов машинного обучения.

Основная часть

Поиск и выбор набора данных для построения моделей машинного обучения.

Описание выбранного датасета

Выбранный набор данных описывает выборку звёзд-кандидатов в пульсары, собранную во время исследования «The High Time Resolution Universe Survey».

Информация об атрибутах:

Каждый кандидат описывается 8 непрерывными переменными и одной переменной класса. Первые четыре являются простыми статистическими данными, полученными из интегрированного импульсного профиля (сложенного профиля). Это массив непрерывных переменных, описывающих разрешенную по долготе версию сигнала, которая была усреднена как по времени, так и по частоте . Остальные четыре переменные аналогично получены из кривой DM-SNR.

- Среднее значение интегрального профиля.
- Стандартное отклонение интегрированного профиля.
- Избыточный эксцесс интегрального профиля.
- Асимметрия интегрированного профиля.
- Среднее значение кривой DM-SNR.
- Стандартное отклонение кривой DM-SNR.
- Избыточный эксцесс кривой DM-SNR.
- Асимметрия кривой DM-SNR.
- Класс

Всего примеров 17 898.

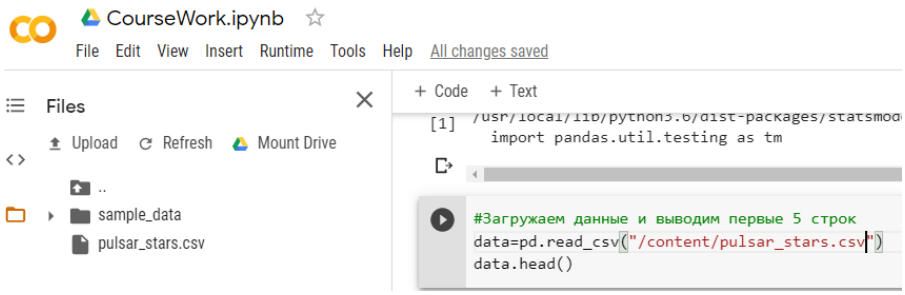
- 1639 положительных примеров.
- 16 259 негативных примеров.

В рассматриваемом примере будем решать задачу классификации. Для классификации в качестве целевого признака будем использовать "target_class" (Класс). Поскольку признак содержит только значения 0 и 1, 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
%matplotlib inline
sns.set(style="ticks")
```

Загрузка данных.
Загрузим файл датасета в помощью библиотеки Pandas.



Проведем разведочный анализ. Построим графики, необходимые для понимания структуры данных. Анализируем и заполненям пропуски в данных.

Основные характеристики датасета

```
#Загружаем данные и выводим первые 5 строк
data=pd.read_csv("/content/pulsar_stars.csv")
data.head()
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0

```
# Размер  датасета
data.shape
```

```
(17898, 9)
```

```
# Список колонок
data.columns
```

```
Index([' Mean of the integrated profile',
       ' Standard deviation of the integrated profile',
       ' Excess kurtosis of the integrated profile',
       ' Skewness of the integrated profile', ' Mean of the DM-SNR curve',
       ' Standard deviation of the DM-SNR curve',
       ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

```
# Список колонок с типами данных
data.dtypes
```

```
 Mean of the integrated profile                float64
 Standard deviation of the integrated profile  float64
 Excess kurtosis of the integrated profile     float64
 Skewness of the integrated profile            float64
 Mean of the DM-SNR curve                      float64
 Standard deviation of the DM-SNR curve        float64
 Excess kurtosis of the DM-SNR curve           float64
 Skewness of the DM-SNR curve                  float64
target_class                                     int64
dtype: object
```

Посмотрим заполненность датасета. Возможно есть пропуски.

```
data.isnull().sum()
```

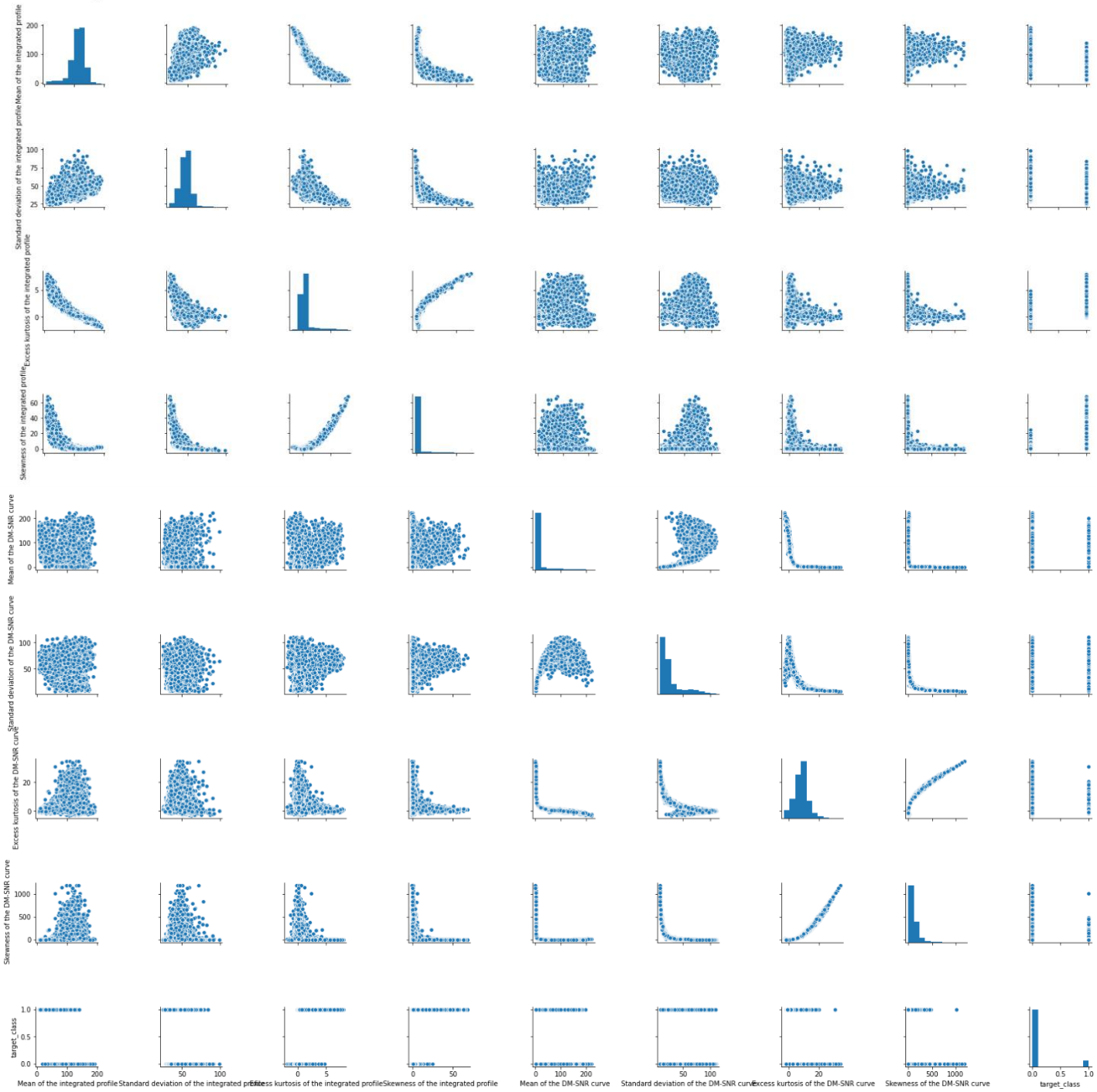
```
 Mean of the integrated profile                0
 Standard deviation of the integrated profile  0
 Excess kurtosis of the integrated profile     0
 Skewness of the integrated profile            0
 Mean of the DM-SNR curve                      0
 Standard deviation of the DM-SNR curve        0
 Excess kurtosis of the DM-SNR curve           0
 Skewness of the DM-SNR curve                  0
target_class                                   0
dtype: int64
```

Пропуски данных отсутствуют

Построим некоторые графики для понимания структуры данных.

```
# Парные диаграммы
sns.pairplot(data)
```

```
<seaborn.axisgrid.PairGrid at 0x7fecac06e828>
```



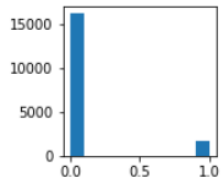
```
#Комбинация гистограмм и диаграмм рассеивания для всего набора данных.
sns.pairplot(data, hue="target_class")
```

```
<seaborn.axisgrid.PairGrid at 0x7fecb0047b38>
```

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

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

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




```
[18] data['target_class'].value_counts()
```

```
0    16259
1     1639
Name: target_class, dtype: int64
```

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

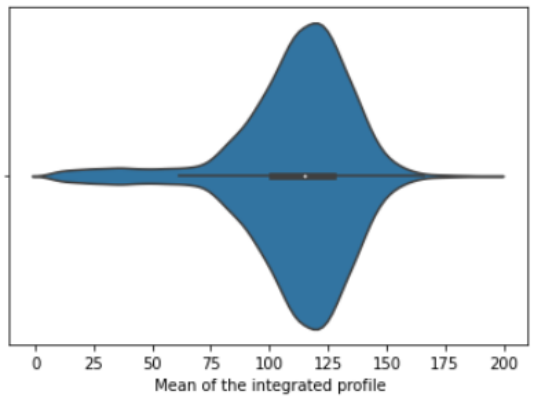
```
Класс 0 составляет 90.84%, а класс 1 составляет 9.16%.
```

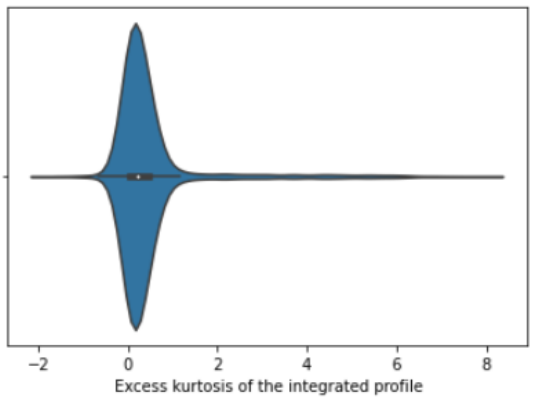
Можем наблюдать явно выраженный дисбаланс классов. Но он является приемлемым.

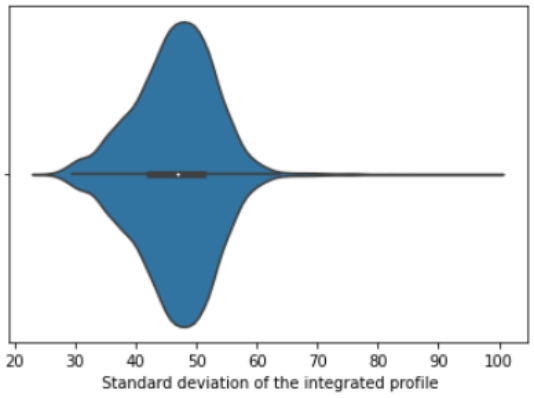
```
data.columns
```

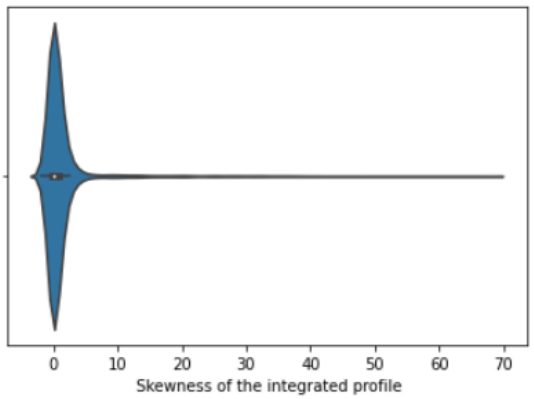
```
Index([' Mean of the integrated profile',
       ' Standard deviation of the integrated profile',
       ' Excess kurtosis of the integrated profile',
       ' Skewness of the integrated profile', ' Mean of the DM-SNR curve',
       ' Standard deviation of the DM-SNR curve',
       ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve',
       'target_class'],
      dtype='object')
```

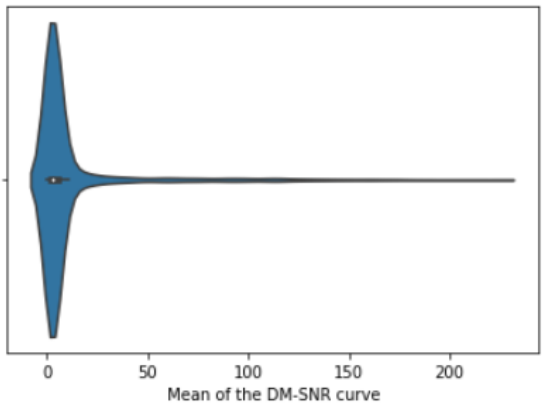
```
[22] # Скрипичные диаграммы для числовых колонок
     for col in [' Mean of the integrated profile',
            ' Standard deviation of the integrated profile',
            ' Excess kurtosis of the integrated profile',
            ' Skewness of the integrated profile', ' Mean of the DM-SNR curve',
            ' Standard deviation of the DM-SNR curve',
            ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve']:
         sns.violinplot(x=data[col])
         plt.show()
```

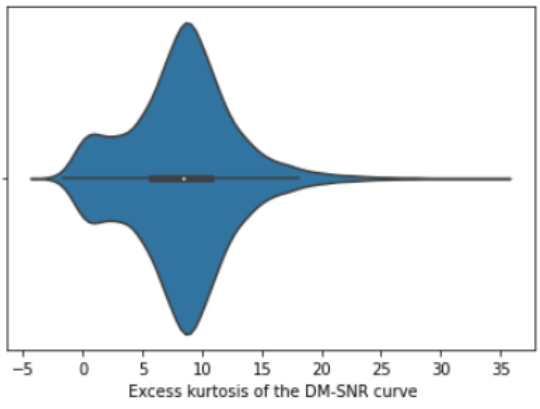


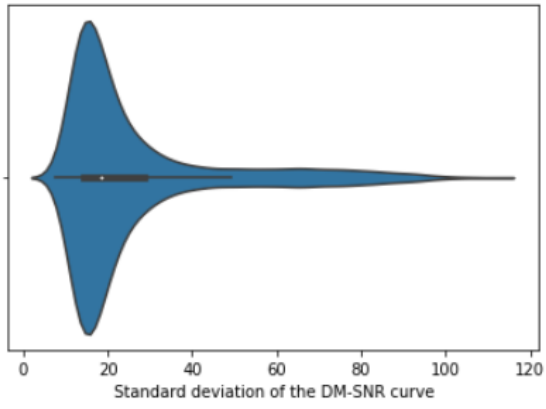


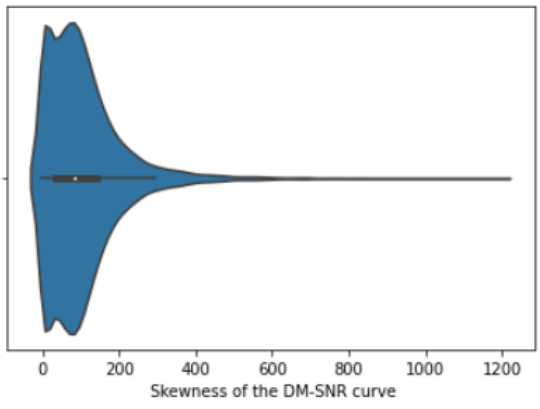












Выбираем признаки, подходящие для построения моделей.

```
data.dtypes
```

```
 Mean of the integrated profile                  float64
 Standard deviation of the integrated profile    float64
 Excess kurtosis of the integrated profile       float64
 Skewness of the integrated profile              float64
 Mean of the DM-SNR curve                        float64
 Standard deviation of the DM-SNR curve          float64
 Excess kurtosis of the DM-SNR curve             float64
 Skewness of the DM-SNR curve                    float64
target_class                                       int64
dtype: object
```

Для построения моделей будем использовать все признаки.

Категориальные признаки отсутствуют, их кодирования не требуется. Исключением является признак target_class, но в представленном датасете он уже закодирован на основе подхода LabelEncoding.

Вспомогательные признаки для улучшения качества моделей строить не будем.

Выполним масштабирование данных.

```
[25] data.head()
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0

```
[26] # Числовые колонки для масштабирования
     scale_cols = [' Mean of the integrated profile',
            ' Standard deviation of the integrated profile',
            ' Excess kurtosis of the integrated profile',
            ' Skewness of the integrated profile', ' Mean of the DM-SNR curve',
            ' Standard deviation of the DM-SNR curve',
            ' Excess kurtosis of the DM-SNR curve', ' Skewness of the DM-SNR curve']
```

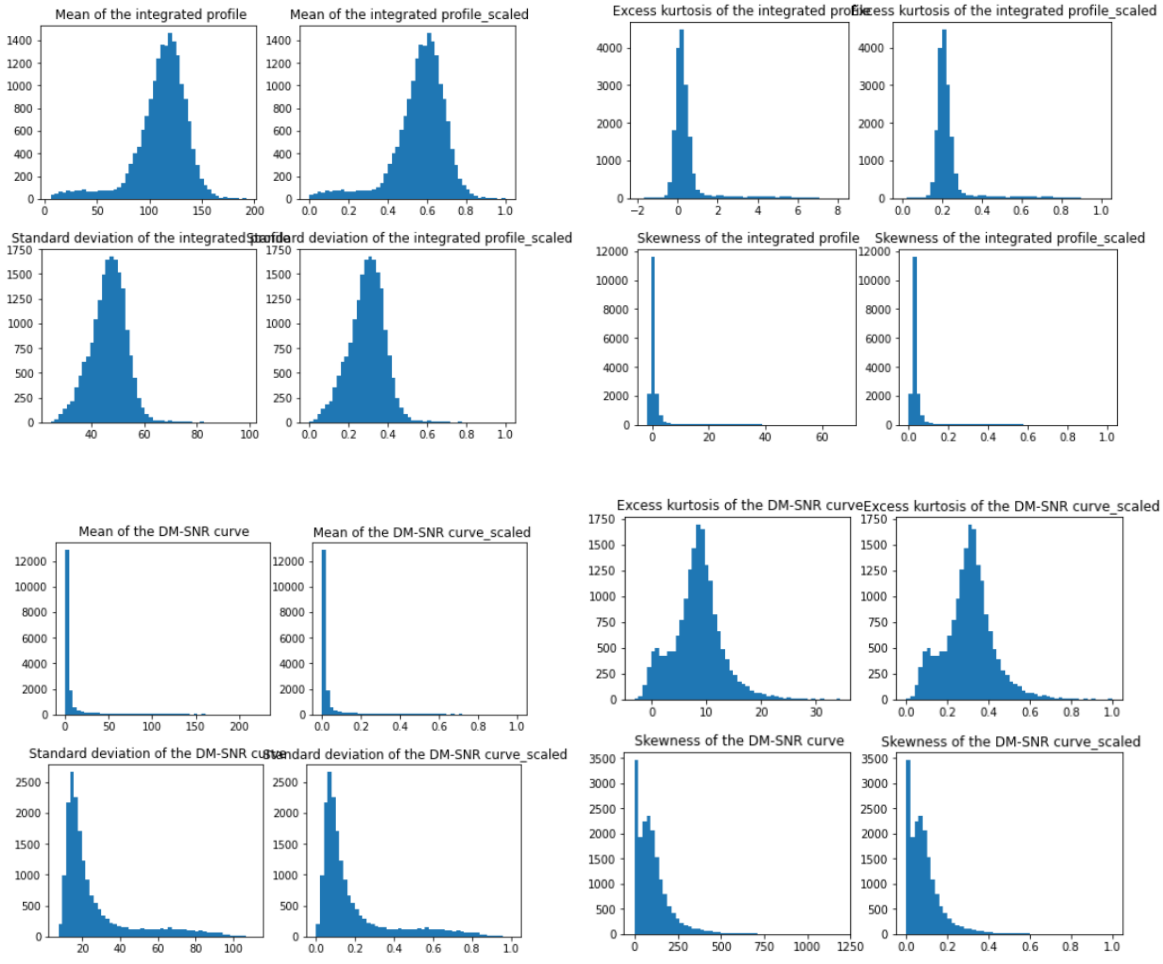
```
[28] from sklearn.preprocessing import MinMaxScaler
     sc1 = MinMaxScaler()
     sc1_data = sc1.fit_transform(data[scale_cols])
```

```
[29] # 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]
```

```
[31] data.head()
```

	Mean of the integrated profile	Standard deviation of the integrated profile	Excess kurtosis of the integrated profile	Skewness of the integrated profile	Mean of the DM-SNR curve	Standard deviation of the DM-SNR curve	Excess kurtosis of the DM-SNR curve	Skewness of the DM-SNR curve	target_class	Mean of the integrated profile_scaled	Standard deviation of the integrated profile_scaled	Excess kurtosis of the integrated profile_scaled	Skewness of the integrated profile_scaled
0	140.562500	55.683782	-0.234571	-0.699648	3.199833	19.110426	7.975532	74.242225	0	0.721342	0.417687	0.165043	0.015627
1	102.507812	58.882430	0.465318	-0.515088	1.677258	14.860146	10.576487	127.393580	0	0.517628	0.460908	0.235415	0.018268
2	103.015625	39.341649	0.323328	1.051164	3.121237	21.744669	7.735822	63.171909	0	0.520346	0.196868	0.221138	0.040677
3	136.750000	57.178449	-0.068415	-0.636238	3.642977	20.959280	6.896499	53.593661	0	0.700933	0.437884	0.181750	0.016534
4	88.726562	40.672225	0.600866	1.123492	1.178930	11.468720	14.269573	252.567306	0	0.443854	0.214847	0.249044	0.041712

```
[32] # 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()
```



Проведем корреляционный анализ данных.

```
[33] corr_cols_1 = scale_cols + ['target_class']
     corr_cols_1
```

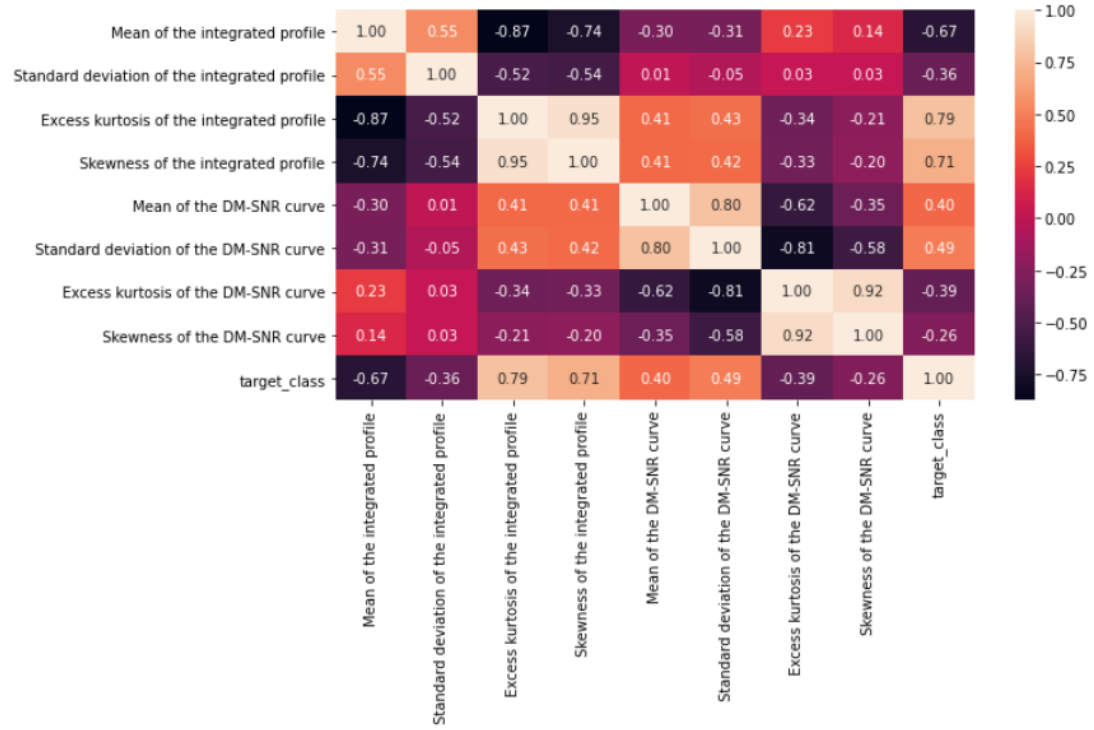
```
[' Mean of the integrated profile',
 ' Standard deviation of the integrated profile',
 ' Excess kurtosis of the integrated profile',
 ' Skewness of the integrated profile',
 ' Mean of the DM-SNR curve',
 ' Standard deviation of the DM-SNR curve',
 ' Excess kurtosis of the DM-SNR curve',
 ' Skewness of the DM-SNR curve',
 'target_class']
```

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

```
[' Mean of the integrated profile_scaled',
 ' Standard deviation of the integrated profile_scaled',
 ' Excess kurtosis of the integrated profile_scaled',
 ' Skewness of the integrated profile_scaled',
 ' Mean of the DM-SNR curve_scaled',
 ' Standard deviation of the DM-SNR curve_scaled',
 ' Excess kurtosis of the DM-SNR curve_scaled',
 ' Skewness of the DM-SNR curve_scaled',
 'target_class']
```

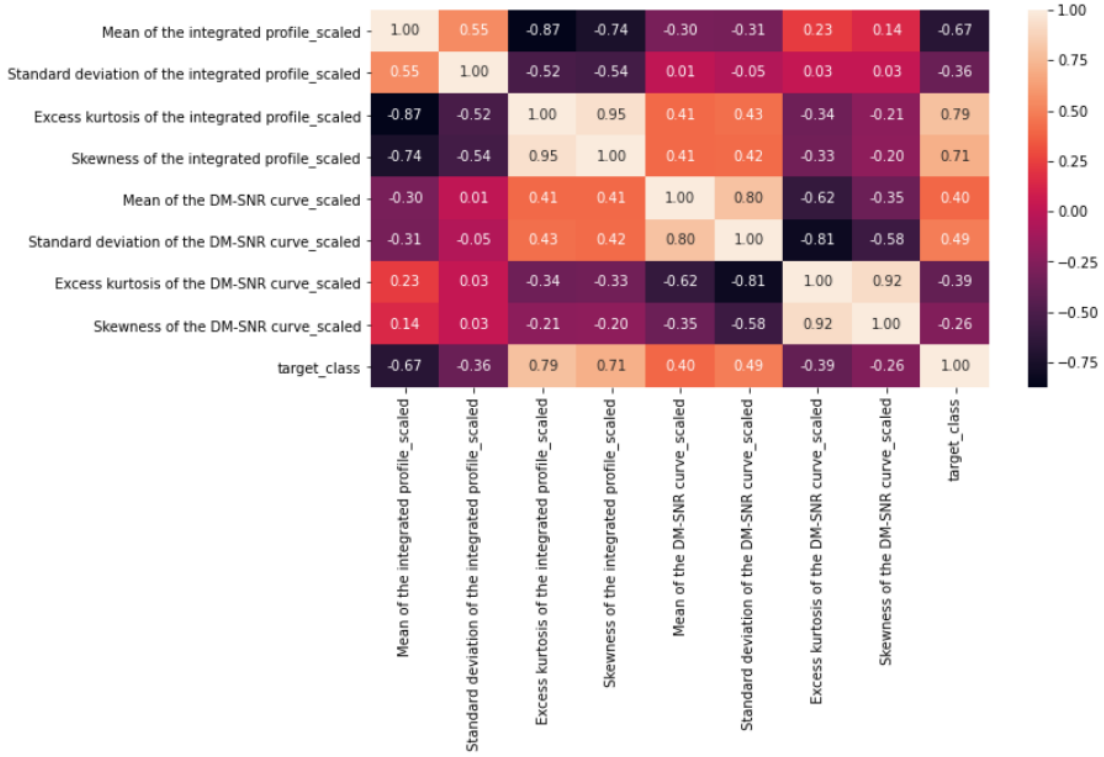
```
[37] fig, ax = plt.subplots(figsize=(10,5))
     sns.heatmap(data[corr_cols_1].corr(), annot=True, fmt='.2f')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feca7a23860>
```



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

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feca6703c88>
```



На основе корреляционной матрицы можно сделать следующие выводы:

1. Корреляционные матрицы для исходных и масштабированных данных совпадают.

2. Целевой признак классификации "target_class" наиболее сильно коррелирует с признаками:
   «Excess kurtosis of the integrated profile» (Избыточный эксцесс интегрального профиля) - 0.79
   «Skewness of the integrated profile» (Асимметрия интегрированного профиля) - - 0.71
   «Standard deviation of the DM-SNR curve» (Стандартное отклонение кривой DM-SNR) - 0.49.
   Эти признаки следует оставить в модели классификации.

3. Признаки «Excess kurtosis of the DM-SNR curve» (Избыточный эксцесс кривой DM-SNR) и «Skewness of the DM-SNR curve» (Асимметрия кривой DM-SNR) имеют корреляцию, близкую по модулю к 1, поэтому оба признака не следуют включать в модели. Следовало бы исключить один из них, но учитывая слишком малую корреляцию их обоих с целевым признаком, можем сделать вывод о необходимости исключения каждого их этих признаков из модели.

4. Признаки «Mean of the integrated profile» (Среднее значение интегрального профиля) и «Standard deviation of the integrated profile» (Стандартное отклонение

интегрированного профиля) 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.

Выбираем метрики 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.

```
[40] 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()
```

Выбираем наиболее подходящие модели 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.

Формируем обучающую и тестовую выборку на основе исходного набора данных.

Имеем масштабированные данные:

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

```
data.columns
```

```
Index([' Mean of the integrated profile_scaled',
       ' Standard deviation of the integrated profile_scaled',
       ' Excess kurtosis of the integrated profile_scaled',
       ' Skewness of the integrated profile_scaled',
       ' Mean of the DM-SNR curve_scaled',
       ' Standard deviation of the DM-SNR curve_scaled',
       ' Excess kurtosis of the DM-SNR curve_scaled',
       ' Skewness of the DM-SNR curve_scaled'],
      dtype='object')
```

```
[62] data.head()
```

	Mean of the integrated profile_scaled	Standard deviation of the integrated profile_scaled	Excess kurtosis of the integrated profile_scaled	Skewness of the integrated profile_scaled	Mean of the DM-SNR curve_scaled	Standard deviation of the DM-SNR curve_scaled	Excess kurtosis of the DM-SNR curve_scaled	Skewness of the DM-SNR curve_scaled
0	0.721342	0.417687	0.165043	0.015627	0.013382	0.113681	0.294986	0.063890
1	0.517628	0.460908	0.235415	0.018268	0.006560	0.072524	0.364015	0.108443
2	0.520346	0.196868	0.221138	0.040677	0.013030	0.139188	0.288624	0.054610
3	0.700933	0.437884	0.181750	0.016534	0.015368	0.131583	0.266348	0.046581
4	0.443854	0.214847	0.249044	0.041712	0.004327	0.039684	0.462029	0.213369

На основе масштабированных данных выделим обучающую и тестовую выборки:

```
[65] # Признаки для задачи классификации
     task_clas_cols = [' Excess kurtosis of the integrated profile_scaled',
            ' Skewness of the integrated profile_scaled',
            ' Mean of the DM-SNR curve_scaled',
            ' Standard deviation of the DM-SNR curve_scaled',]
```

```
[66] # Выборки для задачи классификации
     clas_data = data[task_clas_cols]
```

```
[67] clas_data.head()
```

	Excess kurtosis of the integrated profile_scaled	Skewness of the integrated profile_scaled	Mean of the DM-SNR curve_scaled	Standard deviation of the DM-SNR curve_scaled
0	0.165043	0.015627	0.013382	0.113681
1	0.235415	0.018268	0.006560	0.072524
2	0.221138	0.040677	0.013030	0.139188
3	0.181750	0.016534	0.015368	0.131583
4	0.249044	0.041712	0.004327	0.039684

```
#деление на тестовую и обучающую выборку
clas_X_train, clas_X_test, clas_Y_train, clas_Y_test = train_test_split(
    clas_data, target, test_size=0.2, random_state=1)
clas_X_train.shape, clas_X_test.shape, clas_Y_train.shape, clas_Y_test.shape
```

```
((14318, 4), (3580, 4), (14318,), (3580,))
```

Построим базовое решение (baseline) для выбранных моделей без подбора гиперпараметров

```
[71] from sklearn.linear_model import LogisticRegression
     # Модели
     clas_models = {'LogR': LogisticRegression(),
                    'SVC':SVC(),
                    'Tree':DecisionTreeClassifier(),
                    'RF':RandomForestClassifier(),
                    'GB':GradientBoostingClassifier()}
```

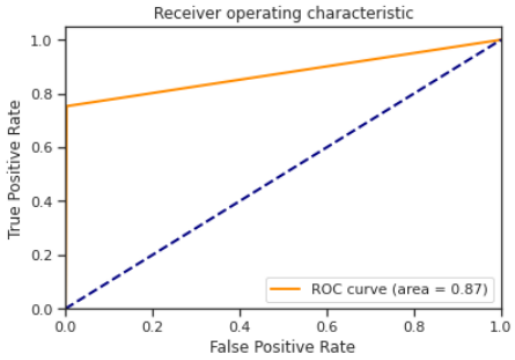
```
[72] # Сохранение метрик
     clasMetricLogger = MetricLogger()
```

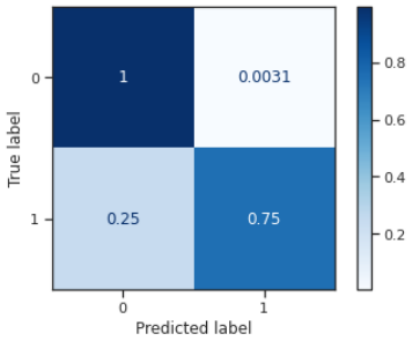
```
[73] def clas_train_model(model_name, model, clasMetricLogger):
         model.fit(clas_X_train, clas_Y_train)
         Y_pred = model.predict(clas_X_test)
         precision = precision_score(clas_Y_test.values, Y_pred)
         recall = recall_score(clas_Y_test.values, Y_pred)
         f1 = f1_score(clas_Y_test.values,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

```
[75] # 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()
```

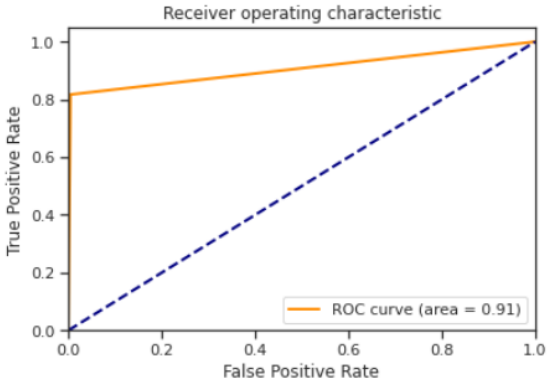
```
for model_name, model in clas_models.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

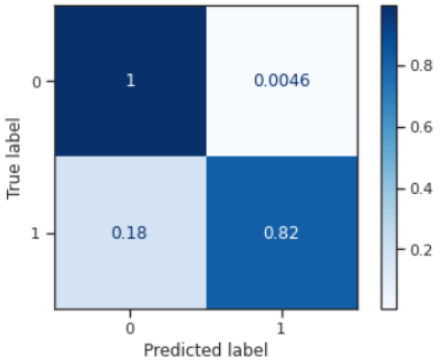
```
**************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None,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



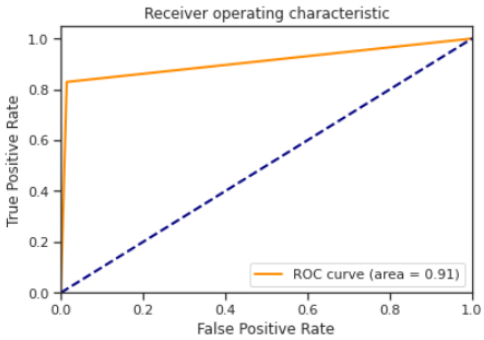


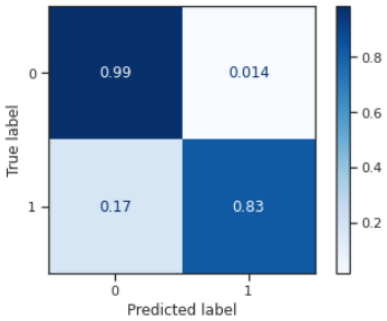

```
**************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)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



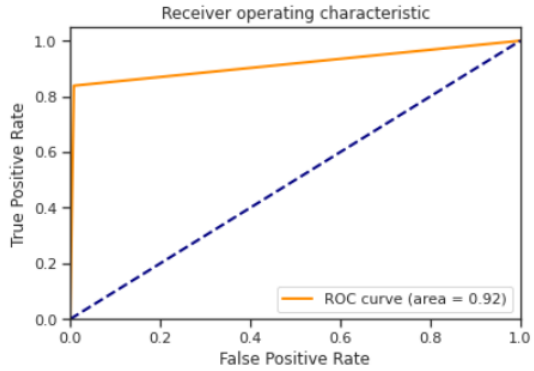


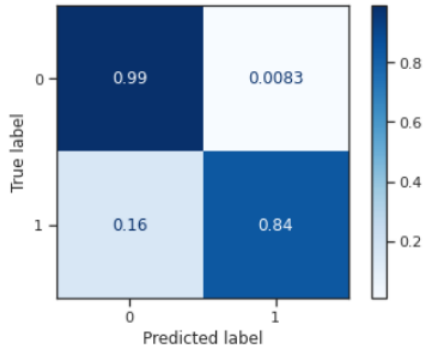
```
**************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                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



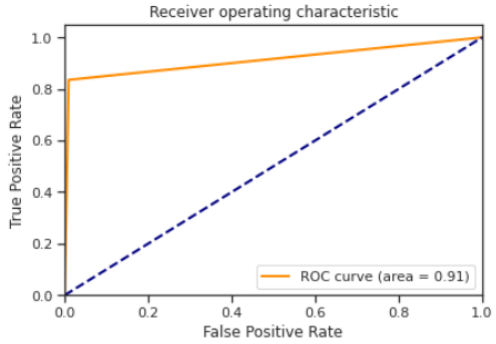


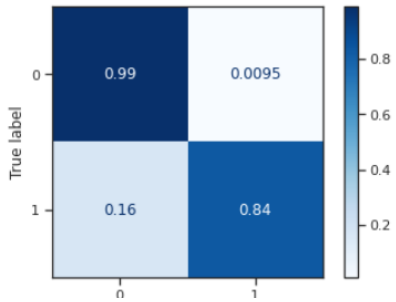

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       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)
**************************************************
```





Подбираем гиперпараметры для выбранных моделей

Кросс-валидация

```
[87] #Кроссвалидация
     scores_log = cross_val_score(LogisticRegression(),
                                  clas_X_train, clas_Y_train, cv=2)
     # Значение метрики accuracy для 2 фолдов
     scores_log, np.mean(scores_log)
```

```
(array([0.96926945, 0.97024724]), 0.9697583461377287)
```

```
[88] scores_svc = cross_val_score(SVC(gamma='auto'),
                                  clas_X_train, clas_Y_train, cv=2)
     # Значение метрики accuracy для 2 фолдов
     scores_svc, np.mean(scores_svc)
```

```
(array([0.9733203 , 0.97359966]), 0.9734599804441961)
```

```
[89] scores_tree = cross_val_score(DecisionTreeClassifier(),
                                  clas_X_train, clas_Y_train, cv=2)
     # Значение метрики accuracy для 2 фолдов
     scores_tree, np.mean(scores_tree)
```

```
(array([0.96773292, 0.96745355]), 0.967593239279229)
```

```
[90] scores_rand_tree = cross_val_score(RandomForestClassifier(),
                                        clas_X_train, clas_Y_train, cv=2)
     # Значение метрики accuracy для 2 фолдов
     scores_rand_tree, np.mean(scores_rand_tree)
```

```
(array([0.97946641, 0.97792988]), 0.9786981421986312)
```

```
[91] scores_boost = cross_val_score(GradientBoostingClassifier(),
                                    clas_X_train, clas_Y_train, cv=2)
     # Значение метрики accuracy для 2 фолдов
     scores_boost, np.mean(scores_boost)
```

```
(array([0.97848862, 0.97471714]), 0.9766028774968571)
```

Подбор гиперпараметров:

```
[139] parameters = {'gamma':[160,150,130,120,110,100,50]}
      clf_gs_svm_svc = GridSearchCV(SVC(), parameters, cv=5, scoring='accuracy')
      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': [160, 150, 130, 120, 110, 100, 50]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

```
[140] # Лучшая модель
      clf_gs_svm_svc.best_estimator_
```

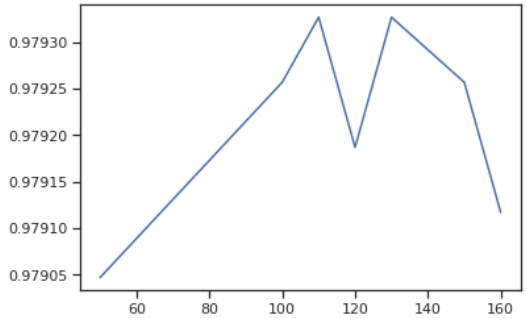
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=110, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
[141] # Лучшее значение параметров
      clf_gs_svm_svc.best_params_
```

```
{'gamma': 110}
```

```
# Изменение качества на тестовой выборке в зависимости от параметра
n_range = np.array([160,150,130,120,110,100,50])
plt.plot(n_range, clf_gs_svm_svc.cv_results_['mean_test_score'])
```

```
[<matplotlib.lines.Line2D at 0x7feca85d0828>]
```



```
[175] parameters = {'penalty':['l1', 'l2', 'elasticnet']}
      clf_gs_log = GridSearchCV(LogisticRegression(), parameters, cv=5, scoring='accuracy')
      clf_gs_log.fit(clas_X_train, clas_Y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:536: FitFai
ValueError: Solver lbfgs supports only 'l2' or 'none' penalties, got l1 penalty.

  FitFailedWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:536: FitFai
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={'penalty': ['l1', 'l2', 'elasticnet']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

```
[176] # Лучшая модель
      clf_gs_log.best_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)
```

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

```
{'penalty': 'l2'}
```

```
[143] 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='accuracy')
      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='accuracy', verbose=0)
```

```
[144] # Лучшая модель
      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=10,
                       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': 10}
```

```
[146] parameters_random_forest = {'n_estimators':[1, 3, 5, 6, 7, 8, 10],
                                  '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='accuracy')
      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],
                         'random_state': [0, 2, 4, 6, 8, 10, 15]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

```
[147] # Лучшая модель
      best_random_forest.best_estimator_
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=6, 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=10,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

```
best_random_forest.best_params_
```

```
{'max_depth': 6, 'n_estimators': 10, 'random_state': 0}
```

```
[187] parameters_gradient_boosting = {'n_estimators':[3, 5, 7, 10, 15, 20],
                                      'max_depth':[3, 5, 7, 9, 10, 15]}
      best_gradient_boosting = GridSearchCV(GradientBoostingClassifier(), parameters_gradient_boosting, cv=5, scoring='accuracy')
      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': [3, 5, 7, 9, 10, 15],
                         'n_estimators': [3, 5, 7, 10, 15, 20]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=0)
```

```
[188] # 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=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=15,
                           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)
```

```
[189] best_gradient_boosting.best_params_
```

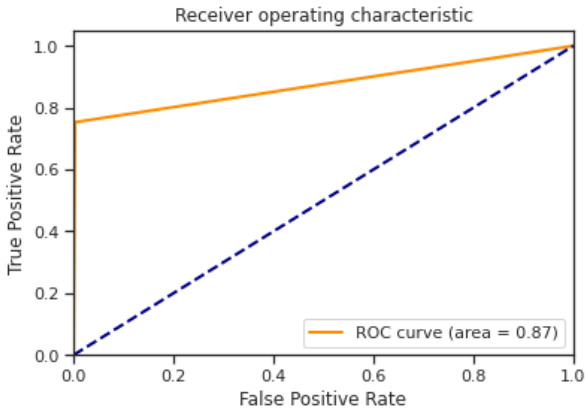
```
{'max_depth': 5, 'n_estimators': 15}
```

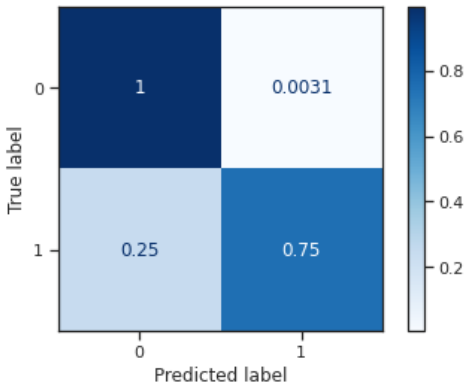
Повторение для найденных оптимальных значений гиперпараметров. Сравнение качества полученных моделей с качеством baseline-моделей.

```
# Новые модели с подобранными гиперпараматерами
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_}
```

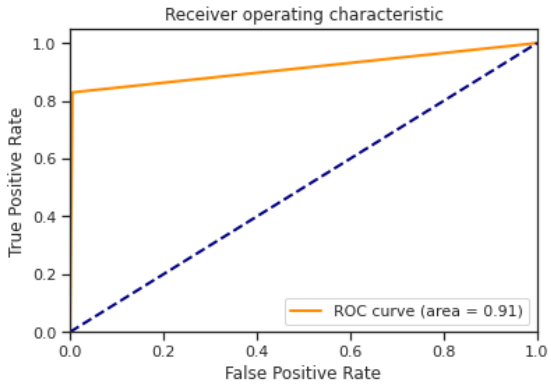
```
for model_name, model in clas_models_grid.items():
    clas_train_model(model_name, model, clasMetricLogger)
```

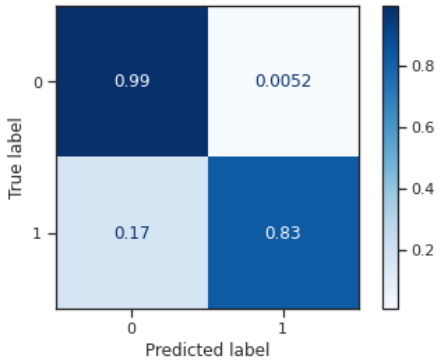
```
******************************************************
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None,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



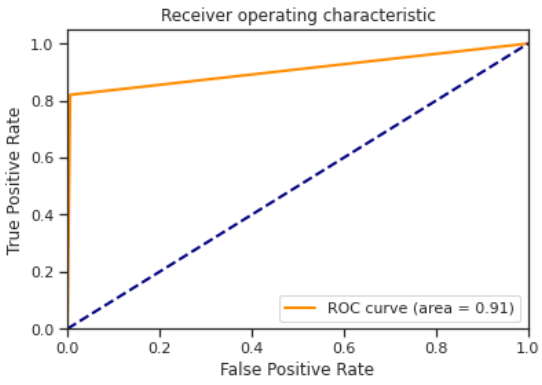


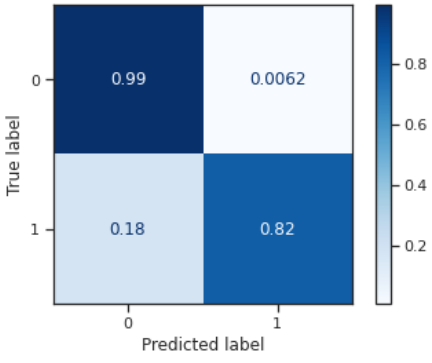
```
*******************************************************
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=110, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
*******************************************************
```



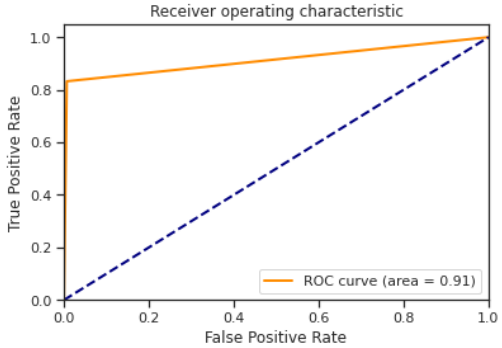


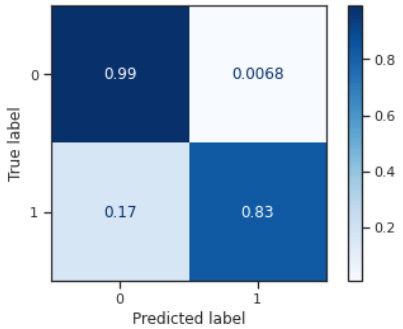

```
*******************************************************
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=10,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
*******************************************************
```



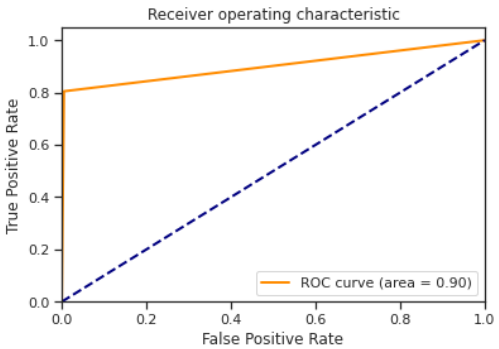


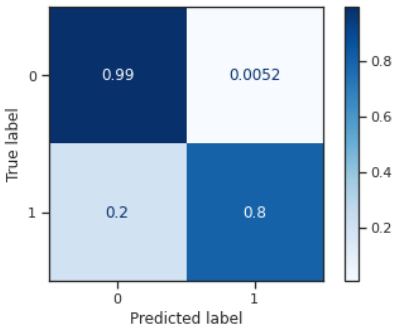
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=6, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
**********************************************************
```






```
**********************************************************
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=15,
                           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)
**********************************************************
```



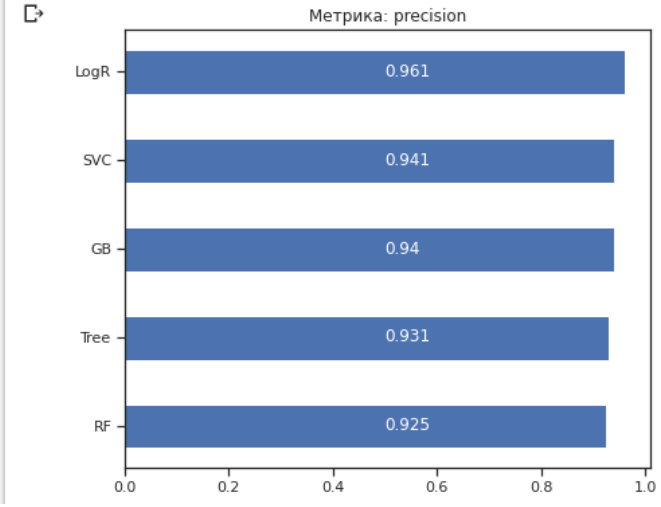


Формируем выводы о качестве построенных моделей на основе выбранных метрик

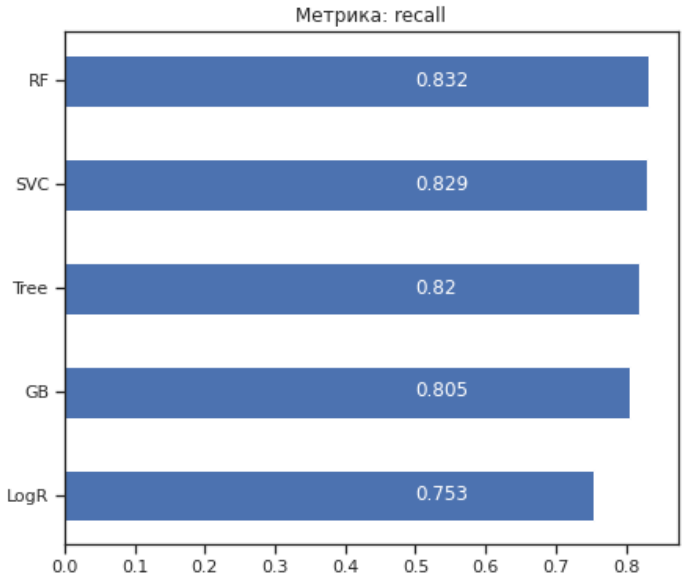
```
[193] # Метрики качества модели
      clas_metrics = clasMetricLogger.df['metric'].unique()
      clas_metrics
```

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

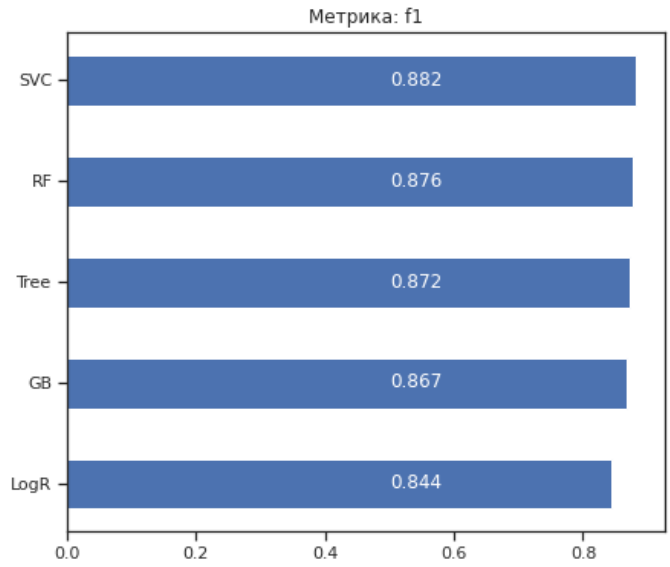
```
# Построим графики метрик качества модели
for metric in clas_metrics:
    clasMetricLogger.plot('Метрика: ' + metric, metric, figsize=(7, 6))
```



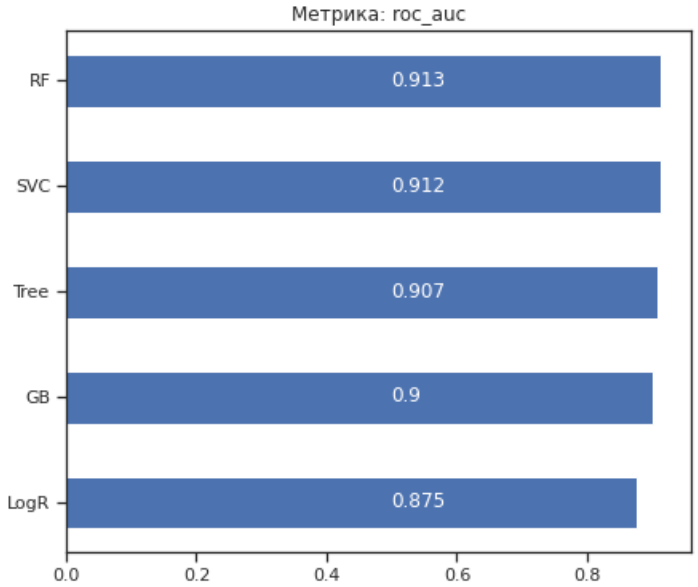
Лучшая модель по метрике precision: Логическая регрессия



Лучшая модель по метрике recall: Случайный лес



Лучшая модель по метрике f1: Метод опорных векторов



Лучшая модель по метрике ROC AUC: Случайный лес

Результат: на основании двух метрик из четырех используемых, лучшей оказалась модель случайного леса. 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.

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: 25.12.2022)
2. Predicting a Pulsar Star [Электронный ресурс]. – Электрон. дан. - URL: https://www.kaggle.com/pavanraj159/predicting-a-pulsar-star (дата обращения: 25.12.2022)
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: 25.12.2022)
4. Scikit learn[Электронный ресурс]. – Электрон. дан. - URL: https://scikit-learn.org/stable/index.html (дата обращения: 25.12.2022)