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»

Отчет по рубежному контролю №2

«Методы построения моделей машинного обучения»

по дисциплине «Технологии машинного обучения»

Вариант №27

Выполнил:
студент группы ИУ5Ц-84Б
Папин А.В.
подпись, дата

Проверил:
к.т.н., доц., Ю.Е. Гапанюк
подпись, дата

2024 г.

СОДЕРЖАНИЕ ОТЧЕТА

1. **Задание**

Для заданного набора данных (по Вашему варианту) постройте модели классификации или регрессии (в зависимости от конкретной задачи, рассматриваемой в наборе данных). Для построения моделей используйте методы 1 и 2 (по варианту для Вашей группы). Оцените качество моделей на основе подходящих метрик качества (не менее двух метрик). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей? Для построения моделей необходимо выполнить требуемую предобработку данных: заполнение пропусков, кодирование категориальных признаков, и т.д.

При решении задач можно выбирать любое подмножество признаков из приведенного набора данных.

Для сокращения времени построения моделей можно использовать фрагмент набора данных (например, первые 200-500 строк).

Методы 1 и 2 для каждой группы приведены в следующей таблице:

Группа	**Метод №1**	**Метод №2**
ИУ5Ц-84Б	Линейная/логистическая регрессия	Градиентный бустинг

Наборы данных:

https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction/data

1. Листинг

1.1. Подключение библиотеки и получение датасета

```
# Уведомление о завершение работы определенного ячейка (очень пригодится для машинного обу
import jupyternotify
%load_ext jupyternotify
```

```
# Подключаем все необходимые библиотеки
import os
import re
import graphviz
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np

# Масштабируемость модели
from sklearn.preprocessing import LabelEncoder, StandardScaler, \
OrdinalEncoder, OneHotEncoder

# Время обучения модели
import timeit
# Вызов библиотеки для отключения предупреждения
import warnings

# Разбиение на обучающую, валидационную и тестовую выборку и кроссвалидацию для повышения
from sklearn.model_selection import train_test_split, GridSearchCV
# Конвейер
from sklearn.pipeline import make_pipeline

# Для машинного обучения
# Ллогическая регрессия (классификация)
from sklearn.linear_model import LogisticRegression
# Бустинги
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.metrics import accuracy_score, precision_score, \
recall_score, f1_score, roc_curve, roc_auc_score
```

```
# Получаем датасет
try:
    df = pd.read_csv('data.csv')
    print('Загружен датасет')
except Exception as ex:
    print('Отсутствует датасет. Проверьте путь файла')
    print('Error:', ex)
```

```
Загружен датасет
```

1.2. Изучение данных

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6819 entries, 0 to 6818
Data columns (total 96 columns):
 #   Column                                                    Non-Null Count  Dtype
---  ------                                                    --------------  -----
 0   Bankrupt?                                                 6819 non-null   int64
 1    ROA(C) before interest and depreciation before interest  6819 non-null   float64
 2    ROA(A) before interest and % after tax                   6819 non-null   float64
 3    ROA(B) before interest and depreciation after tax        6819 non-null   float64
 4    Operating Gross Margin                                   6819 non-null   float64
 5    Realized Sales Gross Margin                              6819 non-null   float64
 6    Operating Profit Rate                                    6819 non-null   float64
 7    Pre-tax net Interest Rate                                6819 non-null   float64
 8    After-tax net Interest Rate                              6819 non-null   float64
 9    Non-industry income and expenditure/revenue              6819 non-null   float64
 10   Continuous interest rate (after tax)                     6819 non-null   float64
 11   Operating Expense Rate                                   6819 non-null   float64
 12   Research and development expense rate                    6819 non-null   float64
```

```
# Привести названия всех колонок к нижнему регистру
df.columns = df.columns.str.lower()
```

```
# Удаляем только первый пробел перед каждым названием столбца
df.columns = df.columns.str.lstrip()
```

```
df.head()
```

	bankrupt?	roa(c) before interest and depreciation before interest	roa(a) before interest and % after tax	roa(b) before interest and depreciation after tax	operating gross margin	realized sales gross margin	operating profit rate	pre-tax net interest rate	after-tax net interest rate	n e
0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.808809	
1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.809301	
2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.808388	
3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.808966	
4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.809304	

5 rows × 96 columns

```
df.tail()
```

	bankrupt?	roa(c) before interest and depreciation before interest	roa(a) before interest and % after tax	roa(b) before interest and depreciation after tax	operating gross margin	realized sales gross margin	operating profit rate	pre-tax net interest rate	after-tax net interest rate
6814	0	0.493687	0.539468	0.543230	0.604455	0.604462	0.998992	0.797409	0.809331
6815	0	0.475162	0.538269	0.524172	0.598308	0.598308	0.998992	0.797414	0.809327
6816	0	0.472725	0.533744	0.520638	0.610444	0.610213	0.998984	0.797401	0.809317
6817	0	0.506264	0.559911	0.554045	0.607850	0.607850	0.999074	0.797500	0.809399
6818	0	0.493053	0.570105	0.549548	0.627409	0.627409	0.998080	0.801987	0.813800

5 rows × 96 columns

1.3. Преобразование данных

```
# Проверим объем занимаемой памяти в Мбайтах до преобразования
print(f'Объем датасета до преобразования: {df.memory_usage(deep=True).sum() / 1024 / 1024:
```

```
Объем датасета до преобразования: 4.995 Мбайт
```

```
original_memory = df.memory_usage(deep=True).sum()
```

```
# Автоматизируем
def change_type_variable(dateframe, show_print_report=False):
    for name_column in dateframe:
        unique_values = dateframe[name_column].unique()

        # if(dateframe[name_column].dtype == 'int64'):
        #     dateframe[name_column] = dateframe[name_column].astype('int32')
        #     if(show_print_report):
        #         print(f'Успешно преобразован тип данных в INT32 для колонки: {name_column}')
        if(dateframe[name_column].dtype == 'float64'):
            dateframe[name_column] = dateframe[name_column].astype('float32')
            if(show_print_report):
                print(f'Успешно преобразован тип данных в FLOAT32 для колонки: {name_column}')
        elif (len(unique_values) == 2 and 0 in unique_values and 1 in unique_values):
                dateframe[name_column] = dateframe[name_column].astype(bool)
                if(show_print_report):
                    print(f'Успешно преобразован тип данных в BOOL для колонки с числами 0 и 1: {name_column}')
        elif(name_column == 'net income flag'):
            dateframe[name_column] = dateframe[name_column].astype(bool)
            if(show_print_report):
                print(f'Успешно преобразован тип данных в BOOL для колонки с числами 0 и 1: {name_column}')
        else:
            pass
    if not(show_print_report):
        print('Все данные успешно преобразованы')
```

```
# Преобразуем их
change_type_variable(df)
```

```
Все данные успешно преобразованы
```

```
# Проверим объем занимаемой памяти в Мбайтах до преобразования
print(f'Объем датасета после преобразования: {df.memory_usage(deep=True).sum() / 1024 / 10
```

```
Объем датасета после преобразования: 2.439 Мбайт
```

```
optimized_memory = df.memory_usage(deep=True).sum()
```

```
# Узнаем, сколько сэкономили памяти
savings_percentage = (original_memory - optimized_memory) / original_memory * 100
print(f"Сэкономлено {savings_percentage:.2f}% памяти")
```

```
Сэкономлено 51.17% памяти
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6819 entries, 0 to 6818
Data columns (total 96 columns):
 #   Column                                                    Non-Null Count  Dtype
---  ------                                                    --------------  -----
 0   bankrupt?                                                 6819 non-null   bool
 1   roa(c) before interest and depreciation before interest   6819 non-null   float32
 2   roa(a) before interest and % after tax                    6819 non-null   float32
 3   roa(b) before interest and depreciation after tax         6819 non-null   float32
 4   operating gross margin                                    6819 non-null   float32
 5   realized sales gross margin                               6819 non-null   float32
 6   operating profit rate                                     6819 non-null   float32
 7   pre-tax net interest rate                                 6819 non-null   float32
 8   after-tax net interest rate                               6819 non-null   float32
 9   non-industry income and expenditure/revenue               6819 non-null   float32
 10  continuous interest rate (after tax)                      6819 non-null   float32
 11  operating expense rate                                    6819 non-null   float32
 12  research and development expense rate                     6819 non-null   float32
```

Рассмотрим описательную статистику

```
df.describe()
```

	roa(c) before interest and depreciation before interest	roa(a) before interest and % after tax	roa(b) before interest and depreciation after tax	operating gross margin	realized sales gross margin	operating profit rate	pre-tax net interest rate	after-tax net interest rate	non-industry income and expenditure/revenue	continuous interest rate (after tax)	...	current liability to current assets	net income to total assets	total assets to gnp price	no-credit interval	gross profit to sales	net income to stockholder's equity	liability to equity
count	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	6819.000000	...	6819.000000	6819.000000	6.819000e+03	6819.000000	6819.000000	6819.000000	6819.000000
mean	0.505180	0.558625	0.553589	0.607948	0.607929	0.998755	0.797190	0.809084	0.303623	0.781381	...	0.031506	0.807760	1.862942e+07	0.623915	0.607946	0.840402	0.280365
std	0.060686	0.065620	0.061595	0.016934	0.016916	0.013010	0.012869	0.013601	0.011163	0.012679	...	0.030845	0.040332	3.764548e+08	0.012289	0.016934	0.014523	0.014463
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000	0.000000
25%	0.476527	0.535543	0.527277	0.600445	0.600434	0.998969	0.797386	0.809312	0.303466	0.781567	...	0.018034	0.796750	9.036205e-04	0.623636	0.600443	0.840115	0.276944
50%	0.502706	0.559802	0.552278	0.605998	0.605976	0.999022	0.797464	0.809375	0.303526	0.781635	...	0.027597	0.810619	2.085213e-03	0.623879	0.605998	0.841179	0.278778
75%	0.535563	0.589157	0.584105	0.613914	0.613842	0.999094	0.797579	0.809469	0.303585	0.781735	...	0.038375	0.826455	5.269777e-03	0.624168	0.613913	0.842357	0.281449
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	...	1.000000	1.000000	9.820000e+09	1.000000	1.000000	1.000000	1.000000

8 rows × 93 columns

Анализ описательной статистики

- В выборке представлены данные о 6819 компаниях.
- Среди этих компаний 2201 обанкротились (т.е., метка "bankrupt?" равна 1), а 4618 не обанкротились (метка "bankrupt?" равна 0).
- Различные показатели отдачи от активов (ROA) варьируются от 0.37 до 0.57, т.е. операционная прибыль, отношение долга к активам и другие, имеют широкий диапазон значений.
- Коэффициенты операционной прибыли, пред-налоговой чистой процентной ставки и пост-налоговой чистой процентной ставки также имеют широкий диапазон значений.

- Важные финансовые показатели, такие как отношение долга к активам, отношение собственного капитала к обязательствам и степень финансового рычага, также имеют различные значения в выборке.
- Все показатели представлены в нормализованных значениях от 0 до 1, что облегчает их сравнение и анализ.

1.4. Пропущенные значения

```
for col in df.columns:
    if df[col].isna().sum() == 0:
        # print(f'У колонки "{col}" нет пропуски')
        pass
    else:
        print(f'У колонки "{col}" присутствует пропуски, кол-во пропусок: {df[col].isna().
```

Отсутствуют пропущенные значения

1.5. Дублирующие значения

```
# Кол-во дублирующие значения
df.duplicated().sum()
```

```
0
```

Отсутствуют дублирующие значения

1.6. Устранение сильных выбросов

Полное устранние выбросов потребует внимательную и непростую работу, т.к. каждая колонка имеет индивидуальные значения и нужно найти особый подход к ним, поэтому будем устранять только те сильные выбросы, стараясь минимизировать количество выбросов

```python
# Функция для удаления выбросов из каждой колонки датафрейма
def remove_outliers(df, left_quantile=0.05, right_quantile=0.95):
    initial_count = len(df)
    # Создаем копию датафрейма
    filtered_df = df.copy()

    # Отфильтровать только числовые столбцы
    numeric_columns = df.select_dtypes(include=[np.number]).columns

    for column in numeric_columns:
        Q1 = df[column].quantile(left_quantile)
        Q3 = df[column].quantile(right_quantile)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Фильтруем значения в пределах границ выбросов
        filtered_df = filtered_df[(filtered_df[column] >= lower_bound) \
        & (filtered_df[column] <= upper_bound)]

    final_count = len(filtered_df)
    lost_count = initial_count - final_count
    lost_percentage = (lost_count / initial_count) * 100

    print(f"Исходный объем датафрейма: {df.shape}")
    print(f"Количество строк после удаления: {final_count}")
    print(f"Количество потерянных строк: {lost_count}")
    print(f"Процент потерянных данных: {lost_percentage:.2f}%")
    print(f"Объем датафрейма после удаления: {filtered_df.shape}")

    return filtered_df
```

```python
# Применяем функцию для удаления выбросов из каждой колонки датафрейма
filtered_df = remove_outliers(df)
```

```
Исходный объем датафрейма: (6819, 96)
Количество строк после удаления: 4510
Количество потерянных строк: 2309
Процент потерянных данных: 33.86%
Объем датафрейма после удаления: (4510, 96)
```

Не так уж много данных потеряно, приступим дальше

1.7. Машинное обучение

```python
# Здесь будем сохранить результаты машинного обучения
results = pd.DataFrame()

# А это будет счетчтиком для нумерация моеделй
count_model = 0
```

1.8. Деление на обучающий и тестовой выборки

Поскольку у нас данные не нормализованы, т.е. на первый взгляд кажется, что значения находится в диапазонах от 0 до 1, и нет необходимости масштабировать признаков, однако существуют колонки, которые не нормализованы

```python
# Создаем список для хранения названий колонок, которые не нормализованы
non_normalized_columns = []

# Проходим по каждой колонке и проверяем максимальное значение
for column in df.columns:
     # Проверяем, отличается ли максимальное значение от 1
    # if df[column].max() != 1.0 and df[column].min() != 0.0:
    # if df[column].max() != 1:
    if (df[column] < 0).any() or (df[column] > 1).any():
        non_normalized_columns.append(column)

# Выводим названия колонок, которые не нормализованы
print("Колонки, требующие нормализации:")
for name_column in non_normalized_columns:
    print('- ', name_column)
```

```
Колонки, требующие нормализации:
-  operating expense rate
-  research and development expense rate
-  interest-bearing debt interest rate
-  revenue per share (yuan ¥)
-  total asset growth rate
-  net value growth rate
-  current ratio
-  quick ratio
-  total debt/total net worth
-  accounts receivable turnover
-  average collection days
-  inventory turnover rate (times)
-  fixed assets turnover frequency
-  revenue per person
-  allocation rate per person
-  quick assets/current liability
-  cash/current liability
-  inventory/current liability
-  long-term liability to current assets
-  current asset turnover rate
-  quick asset turnover rate
-  cash turnover rate
-  fixed assets to assets
-  total assets to gnp price
```

Как и видим, что есть колонки, поэтому будем масштабировать

```python
# Получаем признак и цель
features = filtered_df.drop('bankrupt?', axis=1)
target = filtered_df['bankrupt?']
```

```python
# Разделим обучающую, валидационную и тестовую выборку, потому что 60% обучащие выборки эт
# - 60% обучающей выборки (features_train, target_train)
# - 40% тестовой выборки (features_test, target_test)

# Разделяем данные на обучающую и остальные (валидационную и тестовую) выборки/
features_train, features_test, target_train, target_test = \
train_test_split(features, target, test_size=0.4, random_state=12345)
```

```
display(features_train.head())
display(features_test.head())
```

	roa(c) before interest and depreciation before interest	roa(a) before interest and % after tax	roa(b) before interest and depreciation after tax	operating gross margin	realized sales gross margin	operating profit rate	pre-tax net interest rate	after-tax net interest rate	non-indus expenditu
4126	0.494808	0.554786	0.548745	0.598971	0.598971	0.998986	0.797396	0.809320	
6689	0.617023	0.645497	0.645002	0.624584	0.624584	0.999323	0.797803	0.809601	
5764	0.488666	0.469854	0.543284	0.604434	0.604434	0.998961	0.797108	0.809051	
6483	0.476040	0.537723	0.527866	0.600484	0.600484	0.998973	0.797399	0.809328	
2221	0.505582	0.541649	0.562021	0.616570	0.616570	0.999013	0.797406	0.809331	

5 rows × 95 columns

	roa(c) before interest and depreciation before interest	roa(a) before interest and % after tax	roa(b) before interest and depreciation after tax	operating gross margin	realized sales gross margin	operating profit rate	pre-tax net interest rate	after-tax net interest rate	non-indus expenditu
416	0.418564	0.391463	0.440173	0.584132	0.583880	0.998626	0.796773	0.808609	
4186	0.538439	0.598615	0.578350	0.610653	0.610639	0.999109	0.797776	0.809594	
691	0.513333	0.558112	0.565394	0.609961	0.609961	0.999027	0.797445	0.809357	
4810	0.524009	0.443633	0.582258	0.599771	0.599771	0.998984	0.797032	0.808980	
2410	0.485156	0.546609	0.531881	0.597616	0.597616	0.998998	0.797416	0.809329	

5 rows × 95 columns

1.9. Обучение модели

Выберем метрику F1-score, Accuracy и ROC-curve

Почему выбрал эти?

F1-score это хорошая метрикая для баланса между точностью и полнотой в задачах классификации, особенно в случаях, когда классы несбалансированы

Accuracy это, очевидно, простая и понятная метрика, которая показывает общую долю правильно классифицированных экземпляров в общем количестве экземпляров

ROC AUC это метрика качества классификатора, которая учитывает все пороговые значения и позволяет оценить способность модели отличать между положительными и отрицательными классами. Например, будет

полезно в тех случаях, когда данные несбалансированы или когда интересует только качество ранжирования классификатора

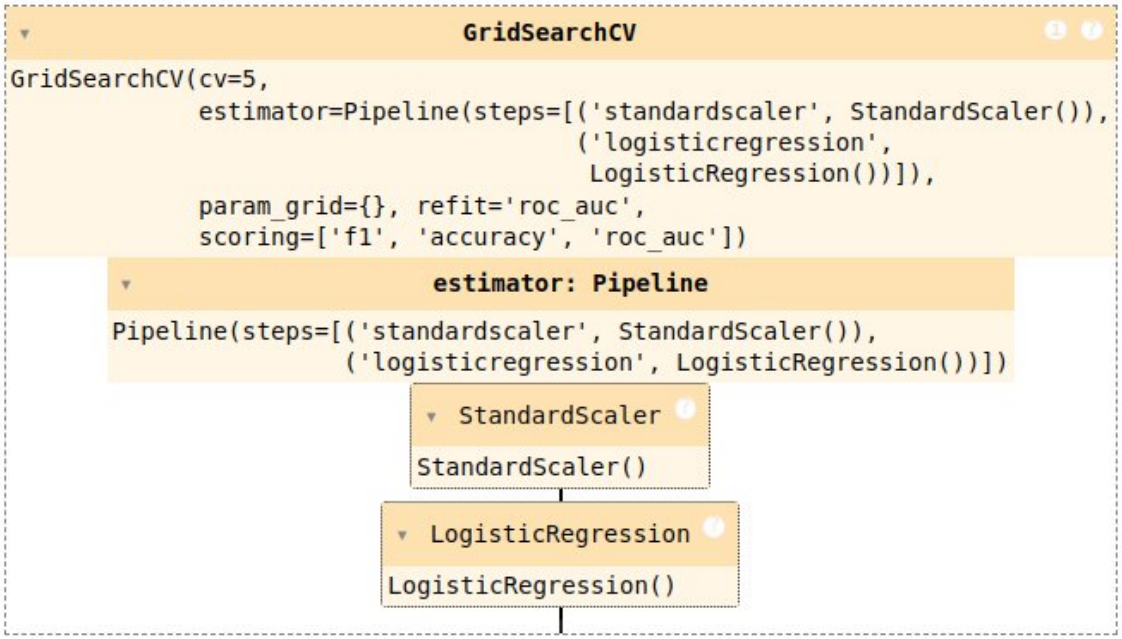
```
metrics = ['f1', 'accuracy', 'roc_auc']
```

```
# Вычисляем все метрики
def compute_metrics(name, target, predictions):
    accuracy = accuracy_score(target, predictions)
    precision = precision_score(target, predictions)
    recall = recall_score(target, predictions)
    f1 = f1_score(target, predictions)

    # Возвращаем результаты
    return {
        f'{name}_accuracy': accuracy,
        f'{name}_precision': precision,
        f'{name}_recall': recall,
        f'{name}_f1': f1,
    }
```

```
# График ROC-кривой
def plot_roc_curve(model, features, target):
    # Получим вероятности для положительного класса
    probs = model.predict_proba(features)[:, 1]

    # Вычислим значения ROC кривой
    fpr, tpr, thresholds = roc_curve(target, probs)

    # Вычислим площадь под ROC кривой (AUC)
    auc = roc_auc_score(target, probs)

    # Построим ROC кривую
    plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
    plt.plot([0, 1], [0, 1], linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend()
    plt.show()
```

1.9.1. LogisticRegression

```
# Устанавливаем нужные параметры
parameters = {}

# Инициализируем модель (включая масштабирование) и GridSearchCV
pipeline_scale = make_pipeline(StandardScaler(), LogisticRegression())
model = GridSearchCV(pipeline_scale, param_grid=parameters, cv=5, scoring=metrics,
                     refit='roc_auc')
display(model)
```




```
%%notify -m f"{pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__}
%%time

# Обучим модель на обучающей выборке
model.fit(features_train, target_train)
time = model.refit_time_
params = model.best_params_

print('TIME TRAIN [s]:', round(time, 2))
```

```
TIME TRAIN [s]: 0.03
CPU times: user 199 ms, sys: 12.2 ms, total: 211 ms
Wall time: 215 ms
Javascript Error: $ is not defined
```

Проверка на тестовой выборки

```
%%time
start_time = timeit.default_timer()

# Получим предсказания на обучающей выборке
train_predictions = model.predict(features_train)

elapsed = round(timeit.default_timer() - start_time, 3)
```

```
CPU times: user 4.9 ms, sys: 8.15 ms, total: 13.1 ms
Wall time: 11.1 ms
```

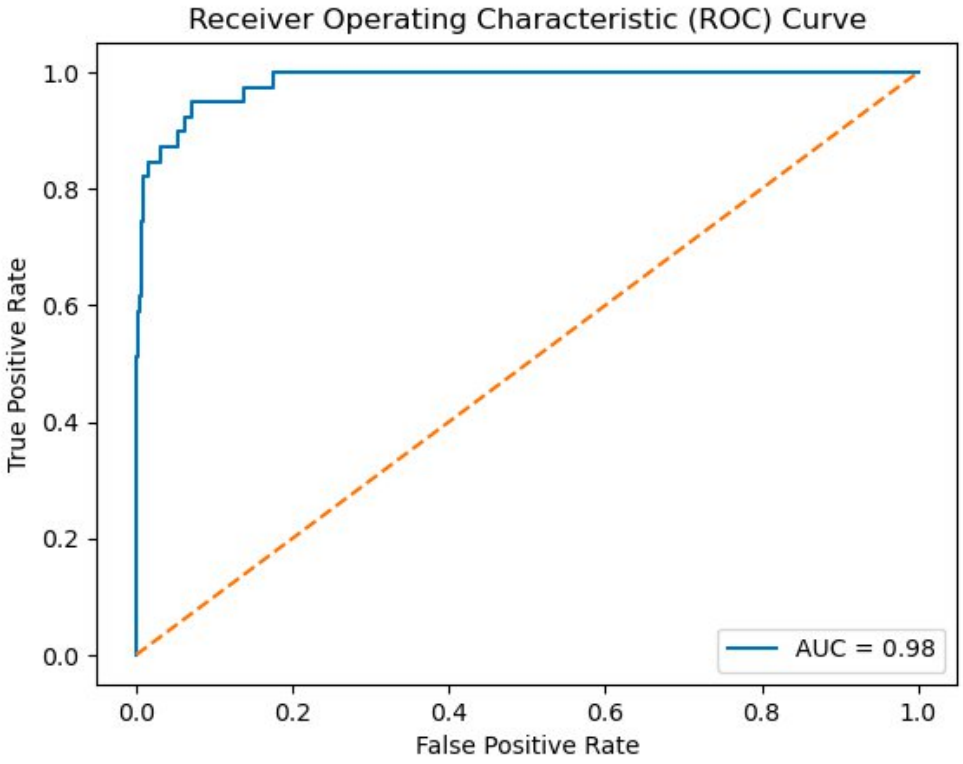
```
%%time
start_time = timeit.default_timer()

# Получим предсказания на тестовой выборки
test_predictions = model.predict(features_test)

elapsed = round(timeit.default_timer() - start_time, 3)
```

```
CPU times: user 5.77 ms, sys: 7.86 ms, total: 13.6 ms
Wall time: 12.1 ms
```

```
# Вызываем функцию для вычисления метрик на обучающей выборке
train_metrics = compute_metrics('TRAIN', target_train, train_predictions)

display(train_metrics)
```

```
{'TRAIN_accuracy': 0.99150036954915,
 'TRAIN_precision': 0.9,
 'TRAIN_recall': 0.46153846153846156,
 'TRAIN_f1': 0.6101694915254238}
```

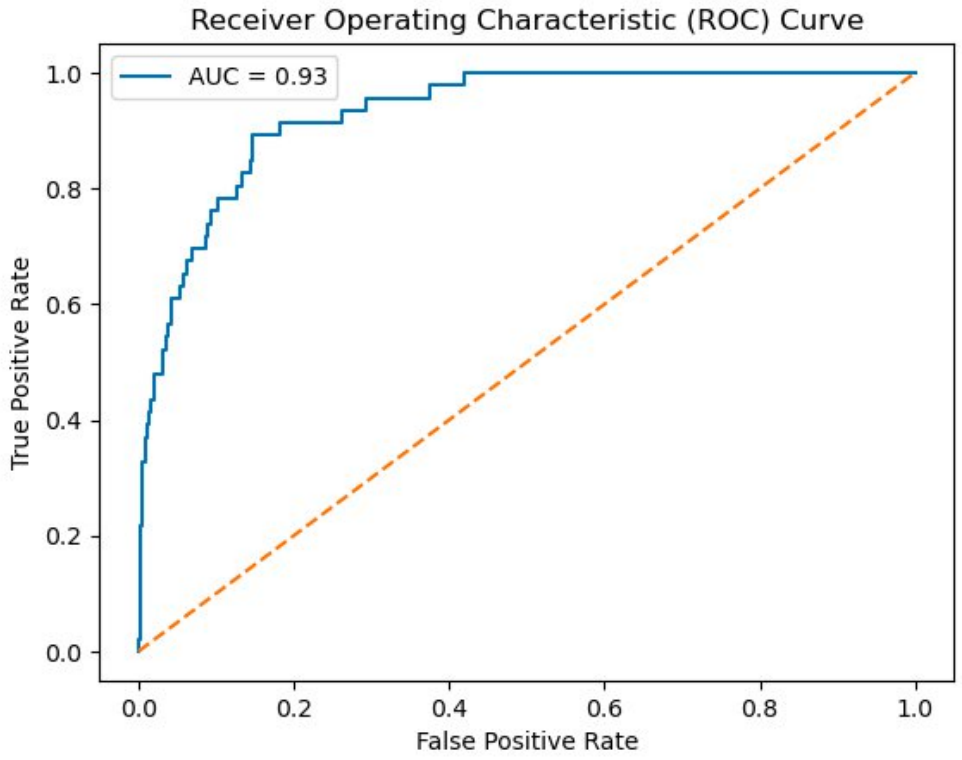
```
plot_roc_curve(model, features_train, target_train)
```




```
# Вызываем функцию для вычисления метрик на тестовой выборке
test_metrics = compute_metrics('TEST', target_test, test_predictions)

display(test_metrics)
```

```
{'TEST_accuracy': 0.9761640798226164,
 'TEST_precision': 0.5555555555555556,
 'TEST_recall': 0.32608695652173914,
 'TEST_f1': 0.410958904109589}
```

```
plot_roc_curve(model, features_test, target_test)
```

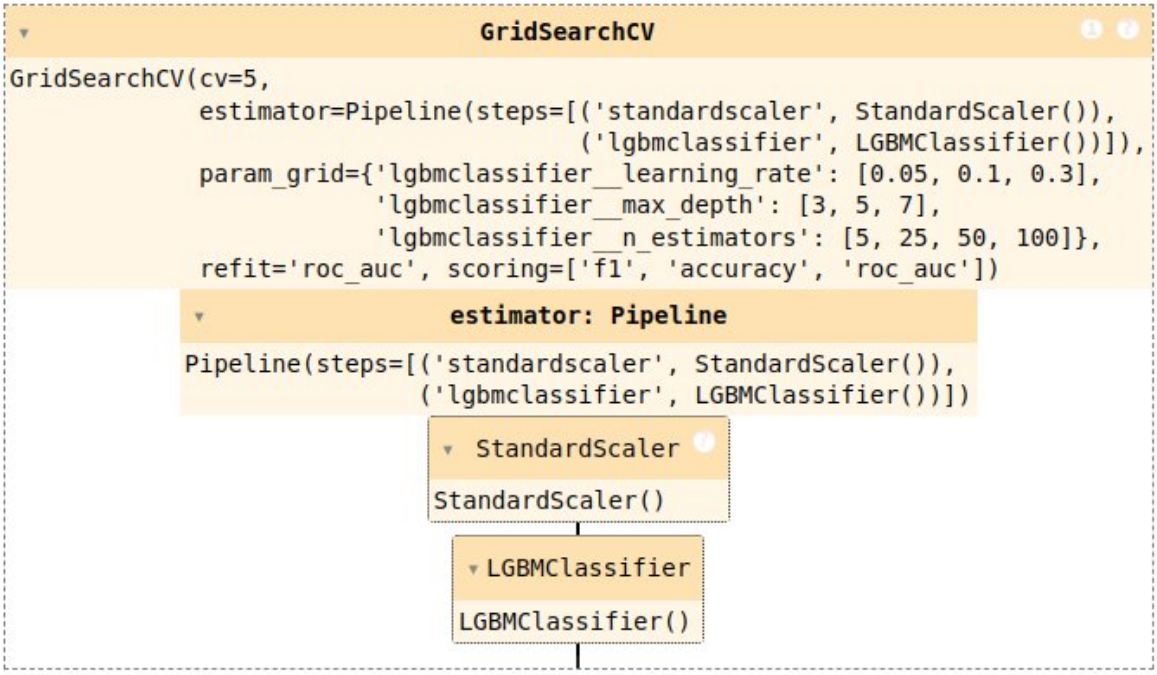



```
# Сохраняем результаты
results[count_model] = pd.Series({
    'NAME': pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__,
    **train_metrics,
    **test_metrics,
    'PREDICTIONS': test_predictions.mean(),
    'TIME TRAINING [s]': model.refit_time_,
    'TIME PREDICTION [s]': elapsed,
    'PARAMETRS': model.best_params_
})

display(results[count_model])
count_model+=1
```

```
NAME                   LogisticRegression
TRAIN_accuracy                     0.9915
TRAIN_precision                       0.9
TRAIN_recall                     0.461538
TRAIN_f1                         0.610169
TEST_accuracy                    0.976164
TEST_precision                   0.555556
TEST_recall                      0.326087
TEST_f1                          0.410959
PREDICTIONS                      0.014967
TIME TRAINING [s]                0.026165
TIME PREDICTION [s]                 0.012
PARAMETRS                              {}
Name: 0, dtype: object
```

1.9.2. LGBMClassifier

```
# Устанавливаем нужные параметры
parameters = {
    # Количество деревьев в модели
    'lgbmclassifier__n_estimators': [5, 25, 50, 100],
     # Скорость обучения модели
    'lgbmclassifier__learning_rate': [0.05, 0.10, 0.30],
     # Максимальная глубина деревьев
    'lgbmclassifier__max_depth': [3, 5, 7],
}

# Инициализируем модель (включая масштабирование) и GridSearchCV
pipeline_scale = make_pipeline(StandardScaler(), LGBMClassifier())
model = GridSearchCV(pipeline_scale, param_grid=parameters, cv=5, scoring=metrics, refit='
display(model)
```



```
%%notify -m f"{pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__}
%%time

# Обучим модель на обучающей выборке
model.fit(features_train, target_train)
time = model.refit_time_
params = model.best_params_

print('TIME TRAIN [s]:', round(time, 2))
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
TIME TRAIN [s]: 0.06
CPU times: user 1min 31s, sys: 1.34 s, total: 1min 33s
Wall time: 11.7 s

Javascript Error: $ is not defined
```

Проверка на тестовой выборки

```
%%time
start_time = timeit.default_timer()

# Получим предсказания на обучающей выборке
train_predictions = model.predict(features_train)

elapsed = round(timeit.default_timer() - start_time, 3)
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^m
ax_depth > num_leaves. (num_leaves=31).
CPU times: user 118 ms, sys: 0 ns, total: 118 ms
Wall time: 14.6 ms
```

```
%%time
start_time = timeit.default_timer()

# Получим предсказания на тестовой выборки
test_predictions = model.predict(features_test)

elapsed = round(timeit.default_timer() - start_time, 3)
```

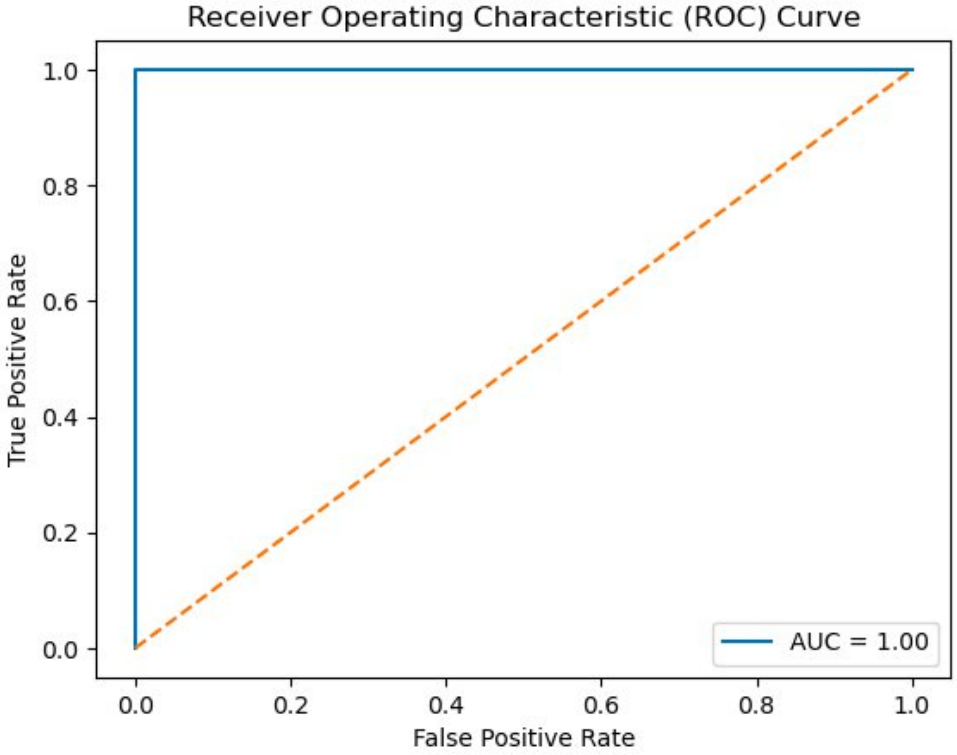
```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^m
ax_depth > num_leaves. (num_leaves=31).
CPU times: user 61.4 ms, sys: 20.1 ms, total: 81.6 ms
Wall time: 10 ms
```

```python
# Вызываем функцию для вычисления метрик на обучающей выборке
train_metrics = compute_metrics('TRAIN', target_train, train_predictions)

display(train_metrics)
```

```
{'TRAIN_accuracy': 0.9977827050997783,
 'TRAIN_precision': 1.0,
 'TRAIN_recall': 0.8461538461538461,
 'TRAIN_f1': 0.9166666666666666}
```

```python
plot_roc_curve(model, features_train, target_train)
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^m
ax_depth > num_leaves. (num_leaves=31).
```

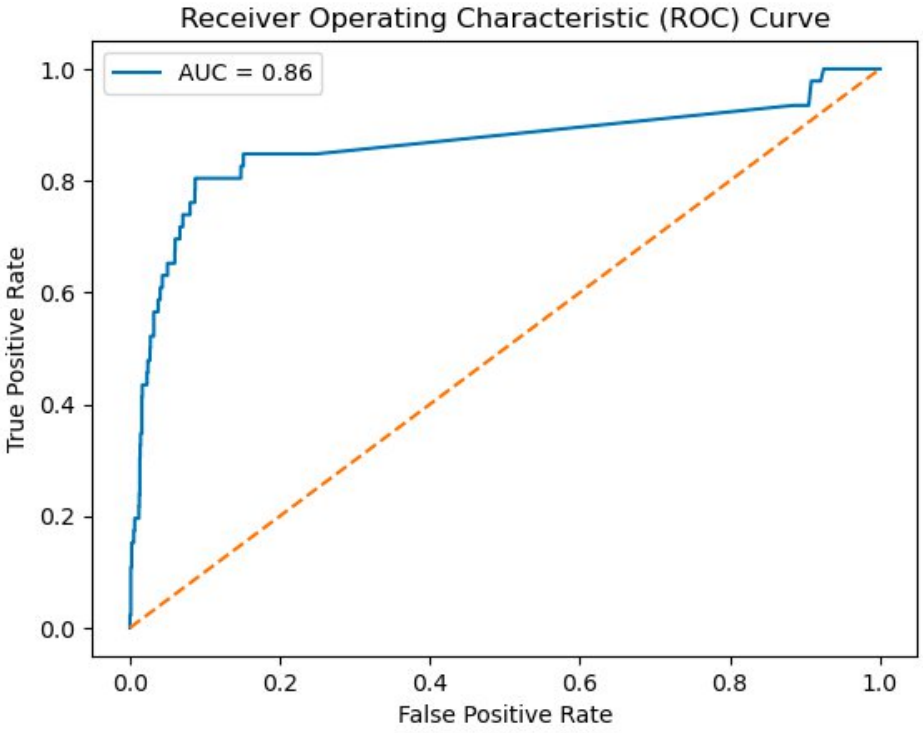


```
# Вызываем функцию для вычисления метрик на тестовой выборке
test_metrics = compute_metrics('TEST', target_test, test_predictions)

display(test_metrics)
```

```
{'TEST_accuracy': 0.975609756097561,
 'TEST_precision': 0.6666666666666666,
 'TEST_recall': 0.08695652173913043,
 'TEST_f1': 0.15384615384615385}
```

```
plot_roc_curve(model, features_test, target_test)
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set num_leaves OR 2^m
ax_depth > num_leaves. (num_leaves=31).
```




```
# Сохраняем результаты
results[count_model] = pd.Series({
    'NAME': pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__,
    **train_metrics,
    **test_metrics,
    'PREDICTIONS': test_predictions.mean(),
    'TIME TRAINING [s]': model.refit_time_,
    'TIME PREDICTION [s]': elapsed,
    'PARAMETRS': model.best_params_
})

display(results[count_model])
count_model+=1
```

```
NAME                                                LGBMClassifier
TRAIN_accuracy                                            0.997783
TRAIN_precision                                                1.0
TRAIN_recall                                              0.846154
TRAIN_f1                                                  0.916667
TEST_accuracy                                              0.97561
TEST_precision                                            0.666667
TEST_recall                                               0.086957
TEST_f1                                                   0.153846
PREDICTIONS                                               0.003326
TIME TRAINING [s]                                         0.057874
TIME PREDICTION [s]                                           0.01
PARAMETRS           {'lgbmclassifier__learning_rate': 0.05, 'lgbmc...
Name: 1, dtype: object
```

1.9.3. CatBoostClassifier

```
# Устанавливаем нужные параметры
parameters = {
    # Количество деревьев в модели
    'catboostclassifier__n_estimators': [5, 25, 50, 100],
     # Скорость обучения модели
    'catboostclassifier__learning_rate': [0.05, 0.10, 0.30],
     # Максимальная глубина деревьев
    'catboostclassifier__max_depth': [3, 5, 7],
}

# Инициализируем модель (включая масштабирование) и GridSearchCV
pipeline_scale = make_pipeline(StandardScaler(), CatBoostClassifier())
model = GridSearchCV(pipeline_scale, param_grid=parameters, cv=5, scoring=metrics,
                     refit='roc_auc')
display(model)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('standardscaler', StandardScaler()),
                                       ('catboostclassifier',
                                        <catboost.core.CatBoostClassifier object
at 0x7fc52f5c2ee0>)]),
             param_grid={'catboostclassifier__learning_rate': [0.05, 0.1, 0.3],
                         'catboostclassifier__max_depth': [3, 5, 7],
                         'catboostclassifier__n_estimators': [5, 25, 50, 100]},
             refit='roc_auc', scoring=['f1', 'accuracy', 'roc_auc'])
```

estimator: Pipeline

```
Pipeline(steps=[('standardscaler', StandardScaler()),
                ('catboostclassifier',
                 <catboost.core.CatBoostClassifier object at 0x7fc52f5c2ee0>)])
```

StandardScaler

```
StandardScaler()
```

CatBoostClassifier

```
<catboost.core.CatBoostClassifier object at 0x7fc52f5c2ee0>
```

```
%%notify -m f"{pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__}
%%time

# Обучим модель на обучающей выборке
model.fit(features_train, target_train)
time = model.refit_time_
params = model.best_params_

print('TIME TRAIN [s]:', round(time, 2))
```

```
86:     learn: 0.0275035        total: 109ms    remaining: 16.3ms
87:     learn: 0.0273325        total: 110ms    remaining: 15ms
88:     learn: 0.0272561        total: 112ms    remaining: 13.8ms
89:     learn: 0.0271507        total: 113ms    remaining: 12.5ms
90:     learn: 0.0270272        total: 114ms    remaining: 11.3ms
91:     learn: 0.0268294        total: 115ms    remaining: 10ms
92:     learn: 0.0266823        total: 116ms    remaining: 8.75ms
93:     learn: 0.0264939        total: 117ms    remaining: 7.5ms
94:     learn: 0.0262839        total: 119ms    remaining: 6.24ms
95:     learn: 0.0261850        total: 120ms    remaining: 5ms
96:     learn: 0.0260808        total: 121ms    remaining: 3.75ms
97:     learn: 0.0258529        total: 123ms    remaining: 2.5ms
98:     learn: 0.0257892        total: 124ms    remaining: 1.25ms
99:     learn: 0.0256728        total: 125ms    remaining: 0us
TIME TRAIN [s]: 0.23
CPU times: user 6min 37s, sys: 27.1 s, total: 7min 4s
Wall time: 40.1 s
```

```
Javascript Error: $ is not defined
```

Проверка на тестовой выборки

```
%%time
start_time = timeit.default_timer()

# Получим предсказания на обучающей выборке
train_predictions = model.predict(features_train)

elapsed = round(timeit.default_timer() - start_time, 3)
```

```
CPU times: user 15.1 ms, sys: 3.93 ms, total: 19 ms
Wall time: 12.6 ms
```

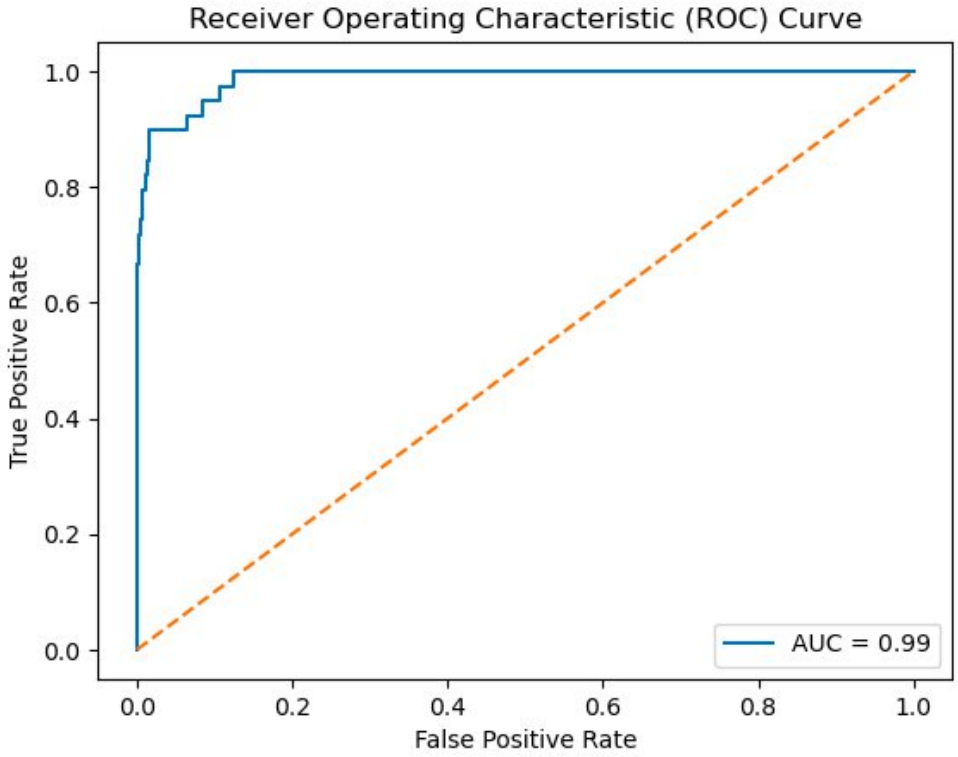
```
%%time
start_time = timeit.default_timer()

# Получим предсказания на тестовой выборки
test_predictions = model.predict(features_test)

elapsed = round(timeit.default_timer() - start_time, 3)
```

```
CPU times: user 14.6 ms, sys: 147 µs, total: 14.7 ms
Wall time: 12 ms
```

```
# Вызываем функцию для вычисления метрик на обучающей выборке
train_metrics = compute_metrics('TRAIN', target_train, train_predictions)

display(train_metrics)
```

```
{'TRAIN_accuracy': 0.9922394678492239,
 'TRAIN_precision': 1.0,
 'TRAIN_recall': 0.46153846153846156,
 'TRAIN_f1': 0.631578947368421}
```

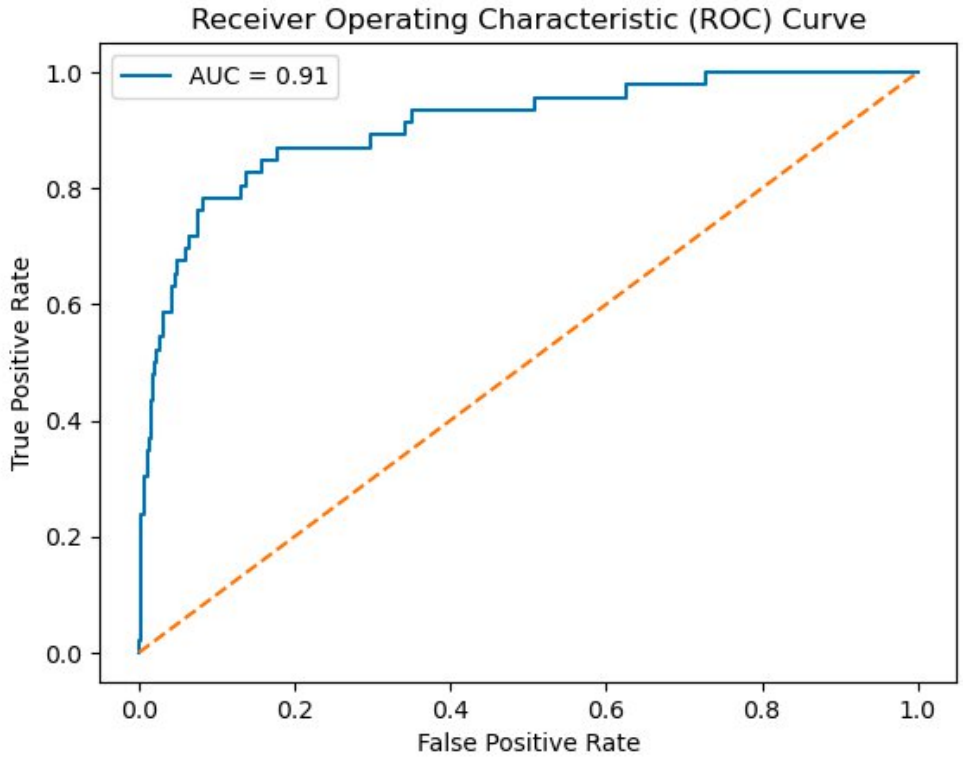
```
plot_roc_curve(model, features_train, target_train)
```




```
# Вызываем функцию для вычисления метрик на тестовой выборке
test_metrics = compute_metrics('TEST', target_test, test_predictions)

display(test_metrics)
```

```
{'TEST_accuracy': 0.9761640798226164,
 'TEST_precision': 0.6,
 'TEST_recall': 0.1956521739130435,
 'TEST_f1': 0.29508196721311475}
```

```
plot_roc_curve(model, features_test, target_test)
```

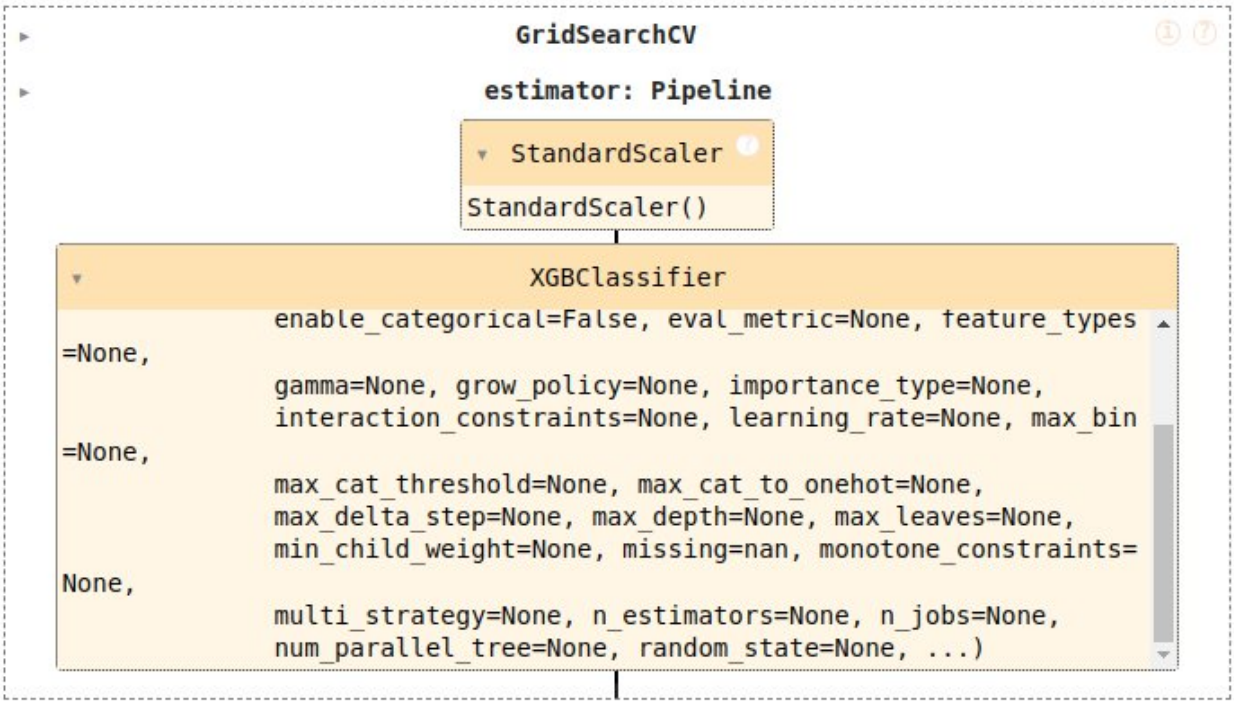



```
# Сохраняем результаты
results[count_model] = pd.Series({
    'NAME': pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__,
    **train_metrics,
    **test_metrics,
    'PREDICTIONS': test_predictions.mean(),
    'TIME TRAINING [s]': model.refit_time_,
    'TIME PREDICTION [s]': elapsed,
    'PARAMETRS': model.best_params_
})

display(results[count_model])
count_model+=1
```

```
NAME                                                CatBoostClassifier
TRAIN_accuracy                                                0.992239
TRAIN_precision                                                    1.0
TRAIN_recall                                                  0.461538
TRAIN_f1                                                      0.631579
TEST_accuracy                                                 0.976164
TEST_precision                                                     0.6
TEST_recall                                                   0.195652
TEST_f1                                                       0.295082
PREDICTIONS                                                   0.008315
TIME TRAINING [s]                                             0.232785
TIME PREDICTION [s]                                              0.012
PARAMETRS             {'catboostclassifier__learning_rate': 0.05, 'c...
Name: 2, dtype: object
```

1.9.4. XGBClassifier

```
# Устанавливаем нужные параметры
parameters = {
    # Количество деревьев в модели
    'xgbclassifier__n_estimators': [5, 10, 30],
     # Скорость обучения модели
    'xgbclassifier__learning_rate': [0.05, 0.10, 0.30],
     # Максимальная глубина деревьев
    'xgbclassifier__max_depth': [3, 5, 7],
}

# Инициализируем модель (включая масштабирование) и GridSearchCV
pipeline_scale = make_pipeline(StandardScaler(), XGBClassifier())
model = GridSearchCV(pipeline_scale, param_grid=parameters, cv=5, scoring=metrics, refit='
display(model)
```




```
%%notify -m f"{pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__}
%%time

# Обучим модель на обучающей выборке
model.fit(features_train, target_train)
time = model.refit_time_
params = model.best_params_

print('TIME TRAIN [s]:', round(time, 2))
```

```
TIME TRAIN [s]: 0.08
CPU times: user 2min 24s, sys: 4.01 s, total: 2min 28s
Wall time: 9.67 s
Javascript Error: $ is not defined
```

Проверка на тестовой выборки

```
%%time
start_time = timeit.default_timer()

# Получим предсказания на обучающей выборке
train_predictions = model.predict(features_train)

elapsed = round(timeit.default_timer() - start_time, 3)
```

```
CPU times: user 209 ms, sys: 2.42 ms, total: 212 ms
Wall time: 14.1 ms
```

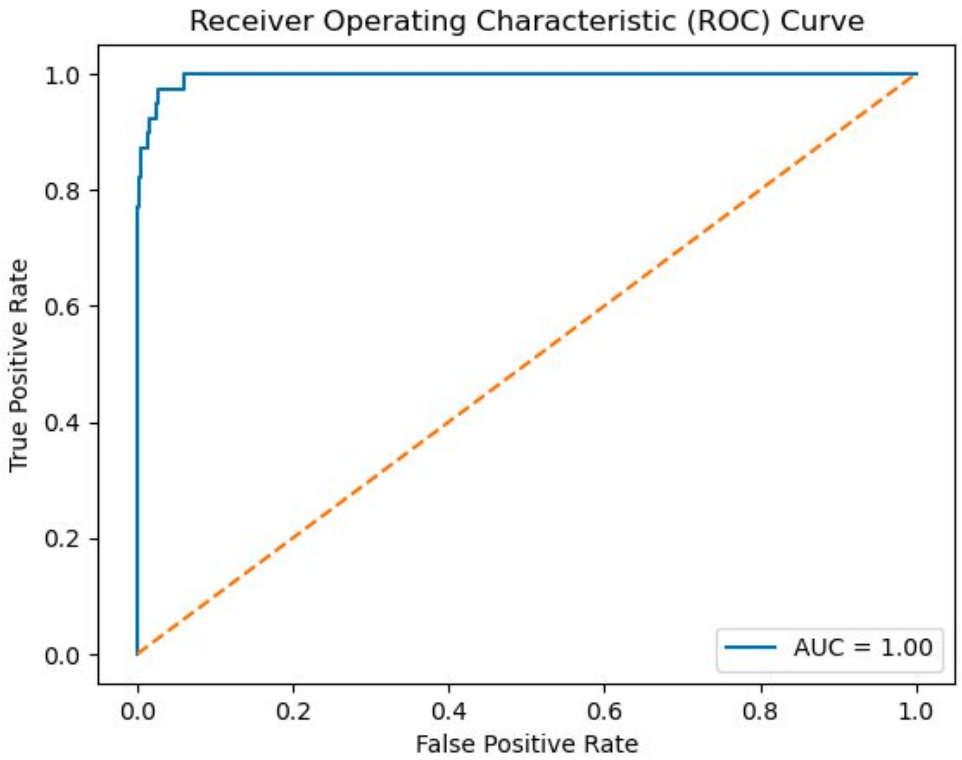
```
%%time
start_time = timeit.default_timer()

# Получим предсказания на тестовой выборки
test_predictions = model.predict(features_test)

elapsed = round(timeit.default_timer() - start_time, 3)
```

```
CPU times: user 296 ms, sys: 4.16 ms, total: 300 ms
Wall time: 28.3 ms
```

```
# Вызываем функцию для вычисления метрик на обучающей выборке
train_metrics = compute_metrics('TRAIN', target_train, train_predictions)

display(train_metrics)
```

```
{'TRAIN_accuracy': 0.991869918699187,
 'TRAIN_precision': 1.0,
 'TRAIN_recall': 0.4358974358974359,
 'TRAIN_f1': 0.6071428571428571}
```

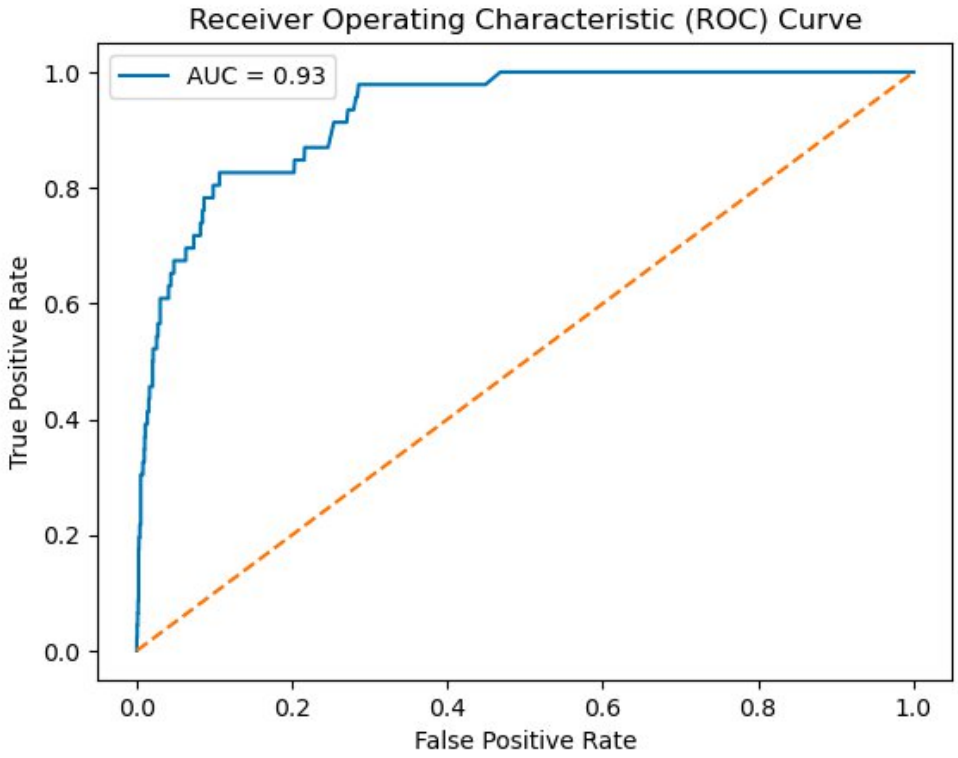
```
plot_roc_curve(model, features_train, target_train)
```



```python
# Вызываем функцию для вычисления метрик на тестовой выборке
test_metrics = compute_metrics('TEST', target_test, test_predictions)

display(test_metrics)
```

```
{'TEST_accuracy': 0.975609756097561,
 'TEST_precision': 0.6,
 'TEST_recall': 0.13043478260869565,
 'TEST_f1': 0.21428571428571427}
```

```python
plot_roc_curve(model, features_test, target_test)
```

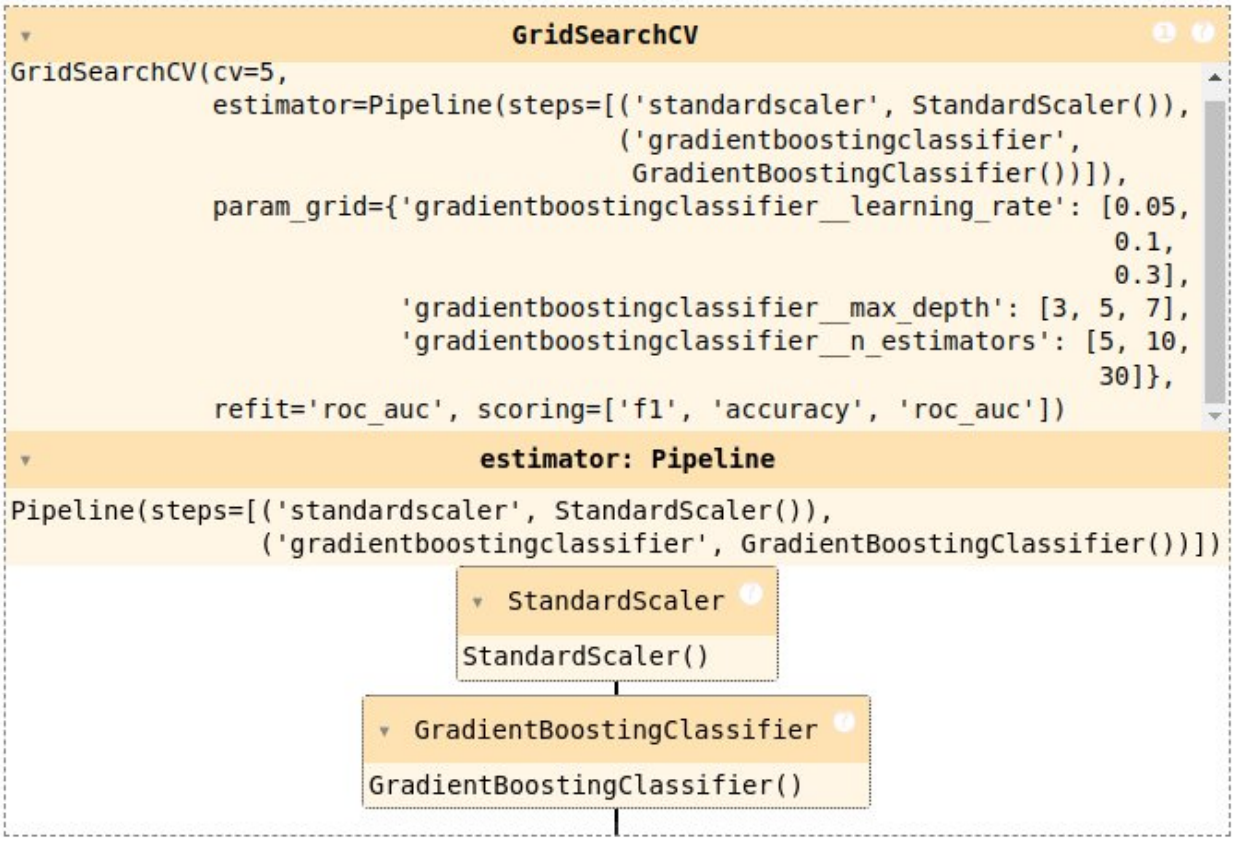



```python
# Сохраняем результаты
results[count_model] = pd.Series({
    'NAME': pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__,
    **train_metrics,
    **test_metrics,
    'PREDICTIONS': test_predictions.mean(),
    'TIME TRAINING [s]': model.refit_time_,
    'TIME PREDICTION [s]': elapsed,
    'PARAMETRS': model.best_params_
})

display(results[count_model])
count_model+=1
```

```
NAME                                                  XGBClassifier
TRAIN_accuracy                                              0.99187
TRAIN_precision                                                 1.0
TRAIN_recall                                               0.435897
TRAIN_f1                                                   0.607143
TEST_accuracy                                               0.97561
TEST_precision                                                  0.6
TEST_recall                                                0.130435
TEST_f1                                                    0.214286
PREDICTIONS                                                0.005543
TIME TRAINING [s]                                          0.079355
TIME PREDICTION [s]                                           0.028
PARAMETRS             {'xgbclassifier__learning_rate': 0.1, 'xgbclas...
Name: 3, dtype: object
```

1.9.5. GradientBoostingClassifier

```
# Устанавливаем нужные параметры
parameters = {
    # Количество деревьев в модели
    'gradientboostingclassifier__n_estimators': [5, 10, 30],
    # Скорость обучения модели
    'gradientboostingclassifier__learning_rate': [0.05, 0.10, 0.30],
    # Максимальная глубина деревьев
    'gradientboostingclassifier__max_depth': [3, 5, 7],
}

# Инициализируем модель (включая масштабирование) и GridSearchCV
pipeline_scale = make_pipeline(StandardScaler(), GradientBoostingClassifier())
model = GridSearchCV(pipeline_scale, param_grid=parameters, cv=5, scoring=metrics, refit='
display(model)
```



```
%%notify -m f"{pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__}
%%time

# Обучим модель на обучающей выборке
model.fit(features_train, target_train)
time = model.refit_time_
params = model.best_params_

print('TIME TRAIN [s]:', round(time, 2))
```

```
TIME TRAIN [s]: 1.87
CPU times: user 2min 44s, sys: 0 ns, total: 2min 44s
Wall time: 2min 44s
Javascript Error: $ is not defined
```

```
%%time
start_time = timeit.default_timer()

# Получим предсказания на обучающей выборке
train_predictions = model.predict(features_train)

elapsed = round(timeit.default_timer() - start_time, 3)
```

```
CPU times: user 13.6 ms, sys: 0 ns, total: 13.6 ms
Wall time: 12.6 ms
```

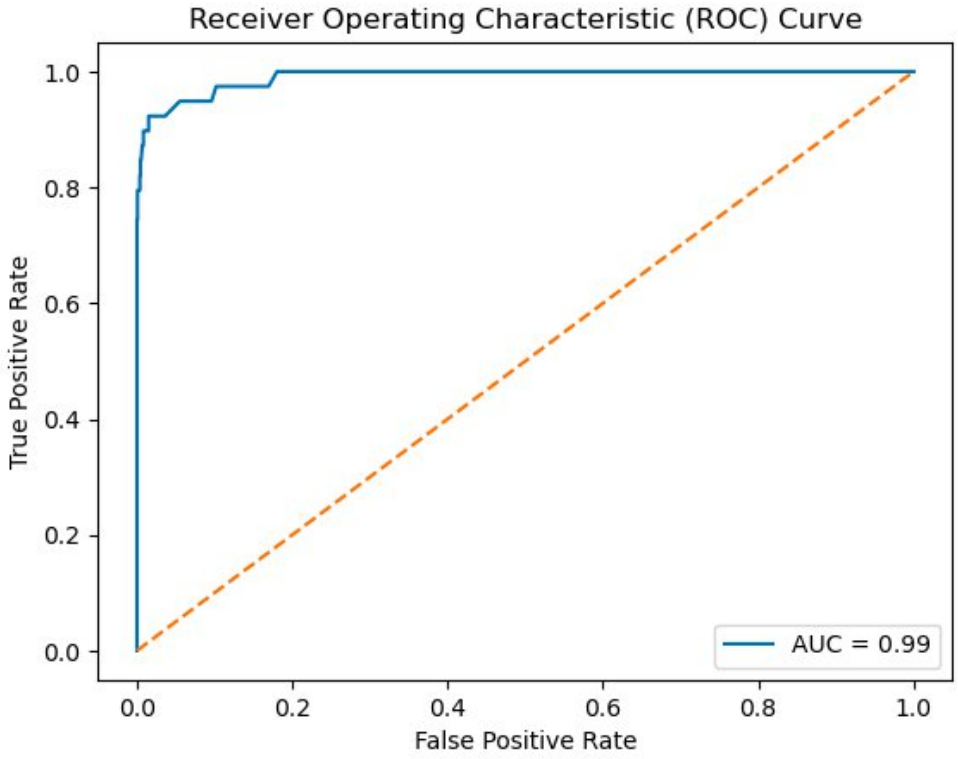
```
%%time
start_time = timeit.default_timer()

# Получим предсказания на тестовой выборки
test_predictions = model.predict(features_test)

elapsed = round(timeit.default_timer() - start_time, 3)
```

```
CPU times: user 12.1 ms, sys: 0 ns, total: 12.1 ms
Wall time: 10.9 ms
```

```
# Вызываем функцию для вычисления метрик на обучающей выборке
train_metrics = compute_metrics('TRAIN', target_train, train_predictions)

display(train_metrics)
```

```
{'TRAIN_accuracy': 0.9940872135994088,
 'TRAIN_precision': 1.0,
 'TRAIN_recall': 0.5897435897435898,
 'TRAIN_f1': 0.7419354838709677}
```

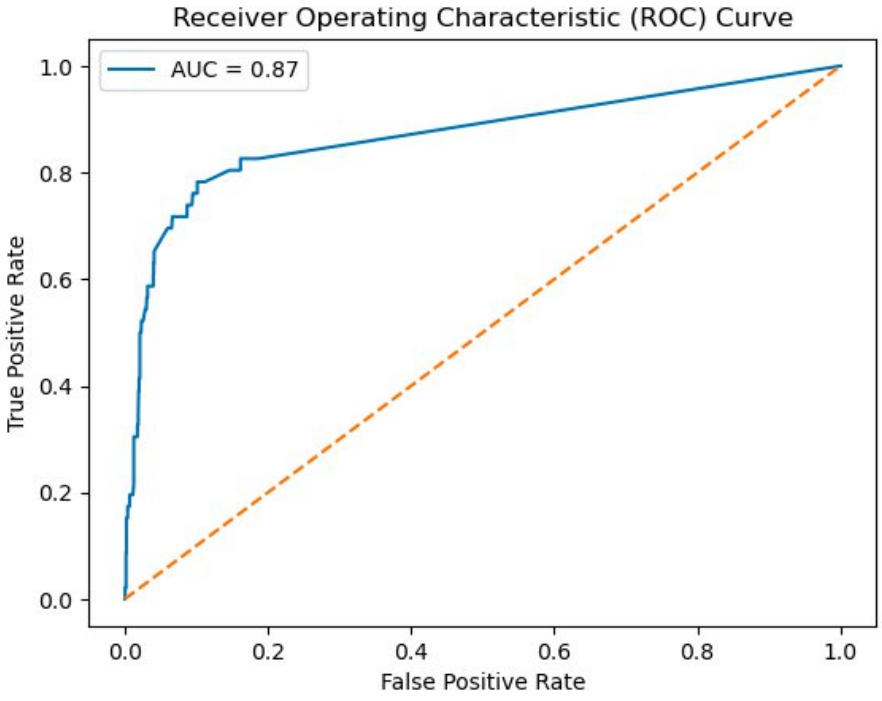
```
plot_roc_curve(model, features_train, target_train)
```




```
# Вызываем функцию для вычисления метрик на тестовой выборке
test_metrics = compute_metrics('TEST', target_test, test_predictions)

display(test_metrics)
```

```
{'TEST_accuracy': 0.9750554323725056,
 'TEST_precision': 0.5384615384615384,
 'TEST_recall': 0.15217391304347827,
 'TEST_f1': 0.23728813559322035}
```

```
plot_roc_curve(model, features_test, target_test)
```



```
# Сохраняем результаты
results[count_model] = pd.Series({
    'NAME': pipeline_scale.named_steps[pipeline_scale.steps[-1][0]].__class__.__name__,
    **train_metrics,
    **test_metrics,
    'PREDICTIONS': test_predictions.mean(),
    'TIME TRAINING [s]': model.refit_time_,
    'TIME PREDICTION [s]': elapsed,
    'PARAMETRS': model.best_params_
})

display(results[count_model])
count_model+=1
```

```
NAME                                                GradientBoostingClassifier
TRAIN_accuracy                                                        0.994087
TRAIN_precision                                                            1.0
TRAIN_recall                                                          0.589744
TRAIN_f1                                                              0.741935
TEST_accuracy                                                         0.975055
TEST_precision                                                        0.538462
TEST_recall                                                           0.152174
TEST_f1                                                               0.237288
PREDICTIONS                                                           0.007206
TIME TRAINING [s]                                                     1.866682
TIME PREDICTION [s]                                                      0.011
PARAMETRS             {'gradientboostingclassifier__learning_rate': ...
Name: 4, dtype: object
```

1.10. Анализ моделей

```
%%notify -m "Total result"
results = pd.DataFrame(results).T
```

```
Javascript Error: $ is not defined
```

```
results
```

	NAME	TRAIN_accuracy	TRAIN_precision	TRAIN_recall	TRAIN_f1	TEST_accuracy	TEST_precision	TEST_recall	TEST_f1	PREDICTIONS	TIME TRAINING [s]	TIME PREDICTION [s]	PARAMETRS
0	LogisticRegression	0.9915	0.9	0.461538	0.610169	0.976164	0.555556	0.326087	0.410959	0.014967	0.026165	0.012	{}
1	LGBMClassifier	0.997783	1.0	0.846154	0.916667	0.97561	0.666667	0.086957	0.153846	0.003326	0.057874	0.01	{'lgbmclassifier__learning_rate': 0.05, 'lgbmc...
2	CatBoostClassifier	0.992239	1.0	0.461538	0.631579	0.976164	0.6	0.195652	0.295082	0.008315	0.232785	0.012	{'catboostclassifier__learning_rate': 0.05, 'c...
3	XGBClassifier	0.99187	1.0	0.435897	0.607143	0.97561	0.6	0.130435	0.214286	0.005543	0.079355	0.028	{'xgbclassifier__learning_rate': 0.1, 'xgbclas...
4	GradientBoostingClassifier	0.994087	1.0	0.589744	0.741935	0.975055	0.538462	0.152174	0.237288	0.007206	1.866682	0.011	{'gradientboostingclassifier__learning_rate': ...

```
# Рассчитываем рейтинг с учетом указанных приоритетов
results['RATING'] = (
    0.5 * (1 - (results['TEST_accuracy'] / results['TEST_accuracy'].max())) +
    0.3 * (1 - (results['TIME TRAINING [s]'] / results['TIME TRAINING [s]'].max())) +
    0.2 * (1 - (results['TIME PREDICTION [s]'] / results['TIME PREDICTION [s]'].max()))
)

# Сортируем DataFrame по убыванию рейтинга
results = results.sort_values(by='RATING', ascending=False)
```

```
results
```

	NAME	TRAIN_accuracy	TRAIN_precision	TRAIN_recall	TRAIN_f1	TEST_accuracy	TEST_precision	TEST_recall	TEST_f1	PREDICTIONS	TIME TRAINING [s]	TIME PREDICTION [s]	PARAMETRS	RATING
1	LGBMClassifier	0.997783	1.0	0.846154	0.916667	0.97561	0.666667	0.086957	0.153846	0.003326	0.057874	0.01	{'lgbmclassifier__learning_rate': 0.05, 'lgbmc...	0.419554
0	LogisticRegression	0.9915	0.9	0.461538	0.610169	0.976164	0.555556	0.326087	0.410959	0.014967	0.026165	0.012	{}	0.410081
2	CatBoostClassifier	0.992239	1.0	0.461538	0.631579	0.976164	0.6	0.195652	0.295082	0.008315	0.232785	0.012	{'catboostclassifier__learning_rate': 0.05, 'c...	0.376874
3	XGBClassifier	0.99187	1.0	0.435897	0.607143	0.97561	0.6	0.130435	0.214286	0.005543	0.079355	0.028	{'xgbclassifier__learning_rate': 0.1, 'xgbclas...	0.287531
4	GradientBoostingClassifier	0.994087	1.0	0.589744	0.741935	0.975055	0.538462	0.152174	0.237288	0.007206	1.866682	0.011	{'gradientboostingclassifier__learning_rate': ...	0.121996

- GradientBoostingClassifier: Показывает идеальную точность на обучающей выборке (1.0), но значительно худшие результаты на

тестовой выборке (0.9684). Это может указывать на переобучение модели.

- LGBMClassifier: Имеет высокую точность на обеих выборках (0.9978 на обучающей и 0.9756 на тестовой). Показывает лучший результат по метрике recall на тестовой выборке (0.8462). Время обучения и прогнозирования относительно невелико.
- CatBoostClassifier: Также показывает высокую точность на обеих выборках (0.9922 на обучающей и 0.9762 на тестовой). Время обучения немного больше, чем у LGBMClassifier.
- LogisticRegression: Имеет неплохую точность на обеих выборках (0.9915 на обучающей и 0.9762 на тестовой). Показывает худшие результаты по метрикам precision и recall на тестовой выборке.
- XGBClassifier: Показывает самую низкую точность на тестовой выборке (0.9756) среди всех моделей, кроме GradientBoostingClassifier.

1.11. Вывод

Удивительно, но LGBMClassifier показала себя лучшей по сравнению с остальными моделями. Да и не говоря о скорости обучения и предсказани. Второе место претендует LogisticRegression, третье место - CatBoostClassifier.

1.12. Гистограмма

```
# Извлечение лучших параметров для логистической регрессии из датафрейма results
best_logistic_params = \
results.loc[results['NAME'] == 'LogisticRegression', 'PARAMETRS'].values[0]

# Создание новой модели логистической регрессии с лучшими параметрами
logistic_regression_new = LogisticRegression(**best_logistic_params)

# Обучение новой модели на обучающем наборе данных
logistic_regression_new.fit(features_train, target_train)
```

```
LogisticRegression
LogisticRegression()
```

```
# Предсказание значений с использованием новой обученной модели
predicted_values_new = logistic_regression_new.predict(features_test)
```

```
# Построение гистограммы для колонки 'bankrupt?' с истинными значениями
plt.hist(target_test.astype(int), bins=2, color='skyblue', edgecolor='black', alpha=0.7, la

# Построение гистограммы для предсказанных значений
plt.hist(predicted_values_new, bins=2, color='red', edgecolor='black', alpha=0.2, label='Пр

plt.xlabel('Банкротство')
plt.ylabel('Частота')
plt.title('Распределение истинных и предсказанных значений "bankrupt?"')
plt.xticks([0, 1], ['Нет', 'Да'])
plt.legend()
plt.show()
```