

## Тестовое задание - анализ данных

В качестве тестового задания требуется построить и валидировать модель для прогноза дефолта по данным в приложении:

- Обучение модели на данных NBKI\_train.csv
- Валидация модели на данных (NBKI\_y\_test.csv, NBKI\_test.csv) Целевая метрика gini

### Формат сдачи:

- ноутбук с исследованием данных и обучением модели и формированием прогноза
- результат валидации модели на тестовых данных в разрезе признака [0]
- дополнительно: скрипт python который запускается из командной строки и на вход принимает параметр –csv файл с тестовыми данными и формирует предсказание вероятности дефолта

## Загрузка данных

Ввод [1]:

```
import pandas as pd
import numpy as np
from scipy import stats

from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.utils.class_weight import compute_class_weight

from catboost import CatBoostClassifier, Pool

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

Ввод [2]:

```
# настройки
# Убираем ограничение отображаемых колонок
pd.set_option("display.max_columns", None)
# Устанавливаем тему по умолчанию
sb_dark = sns.dark_palette('skyblue', 8, reverse=True) # teal
sns.set(palette=sb_dark)
```

Ввод [3]:

```
PATH = "./"
PATH_DATASETS = PATH + "datasets/"
```

Ввод [4]:

```
train_nbki_df = pd.read_csv(PATH_DATASETS + "NBKI_train.csv", index_col="Unnamed: 0")
test_nbki_df = pd.read_csv(PATH_DATASETS + "NBKI_test.csv")
y_test_nbki_df = pd.read_csv(PATH_DATASETS + "NBKI_y_test.csv")
test_nbki_df = pd.merge(test_nbki_df, y_test_nbki_df, on="Unnamed: 0").set_index("Unnamed: 0").reset_index()

train_nbki_df.shape, test_nbki_df.shape, y_test_nbki_df.shape
```

Out[4]:

```
((30007, 134), (29993, 135), (29993, 2))
```

**Ввод [5]:**

```
train_nbki_df.head()
```

**Out[5]:**

106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
0	0	0	0	430	333	5	3	0	0	0	61	40	2	1	0	0	0	0	0	13	1096
0	0	0	0	147	123	3	2	0	0	0	53	46	0	0	0	0	0	1	0	3	123
0	0	0	0	201	167	6	2	0	0	0	24	16	0	0	0	0	0	0	0	5	1826
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	498
0	0	0	0	229	192	2	2	0	0	0	42	35	1	1	0	0	0	0	0	16	182

### Исходные данные

В исходных данных к тестовым данным также предоставлен таргет тестовых данных. Это позволяет использовать тестовые данные для оценки модели. Но есть риск "подгонки" модели именно под этот набор тестовых данных. Это надо учитывать.

## Первичный анализ

**Ввод [6]:**

```
# Проверям типы данных
train_nbki_df.info(verbose=True)
```

116	116	int64
117	117	int64
118	118	int64
119	119	int64
120	120	int64
121	121	int64
122	122	int64
123	123	int64
124	124	int64
125	125	int64
126	126	int64
127	127	int64
128	128	float64
129	129	float64
130	130	float64
131	131	float64
132	132	int64
133	default	float64
		dtypes: float64(11), int64(123)
		memory usage: 30.9 MB

Ввод [7]:

```
# Проверям значения
describe_df = train_nbki_df.describe(include="all")
describe_df
```

Out[7]:

	0	1	2	3	4	5	6	
count	30007.000000	30007.000000	30007.000000	30007.000000	30007.0	3.000700e+04	28819.000000	28918.00
mean	8.053654	1.023894	2.338121	16.326391	10.0	2.124157e+05	46.050036	45.76
std	1.461789	0.266673	1.822649	9.017349	0.0	7.026602e+05	23.971219	23.74
min	1.000000	1.000000	0.000000	1.000000	10.0	0.000000e+00	1.000000	1.00
25%	7.000000	1.000000	1.000000	9.000000	10.0	1.850600e+04	24.000000	24.00
50%	9.000000	1.000000	2.000000	17.000000	10.0	5.000000e+04	50.000000	50.00
75%	9.000000	1.000000	4.000000	24.000000	10.0	1.500000e+05	66.000000	66.00
max	9.000000	4.000000	6.000000	31.000000	10.0	4.072000e+07	99.000000	99.00

Ввод [8]:

```
# Проверяя на мин/макс (также смотрим чтобы не было -inf/inf)
describe_df.loc["min"].min(), describe_df.loc["max"].max()
```

Out[8]:

(−567.0, 606623042.0)

Ввод [9]:

```
# Проверяя кардинальность значений
check_cardinality = train_nbki_df.apply(lambda x: x.nunique())
display(check_cardinality.describe())
print(f"80%: {check_cardinality.quantile(0.80)}")
```

count	134.000000
mean	1073.074627
std	4183.784234
min	1.000000
25%	9.000000
50%	25.000000
75%	78.000000
max	26525.000000
dtype:	float64

80%: 91.4

Ввод [10]:

threshold\_cardinality = 20

## Кардинальность

- Для всей выборки размера 30.000 объектов 75% признаков имеют 80 уникальных значений, а 80% признаков имеют меньше 100 уникальных значений.
- Можем принять, что признаки у которых уникальных значений меньше 20 считаются категориальными.
- Данный порог уникальных значений надо учитывать для дальнейшего анализа

Ввод [11]:

```
# определяем колонки меток и признаков
target_column = "default"
feature_columns = train_nbki_df.columns.drop(target_column)
categorical_columns = list(set(check_cardinality[check_cardinality <= threshold_cardinality].index) - set(feature_columns))
numeric_columns = feature_columns.drop(categorical_columns)

assert len(feature_columns) == len(categorical_columns) + len(numeric_columns), "Ошибка в определении признаков"
assert len(train_nbki_df.columns) == len(feature_columns) + len([target_column]), "Ошибка в определении метки"

len(feature_columns), len(categorical_columns), len(numeric_columns), len([target_column])
```

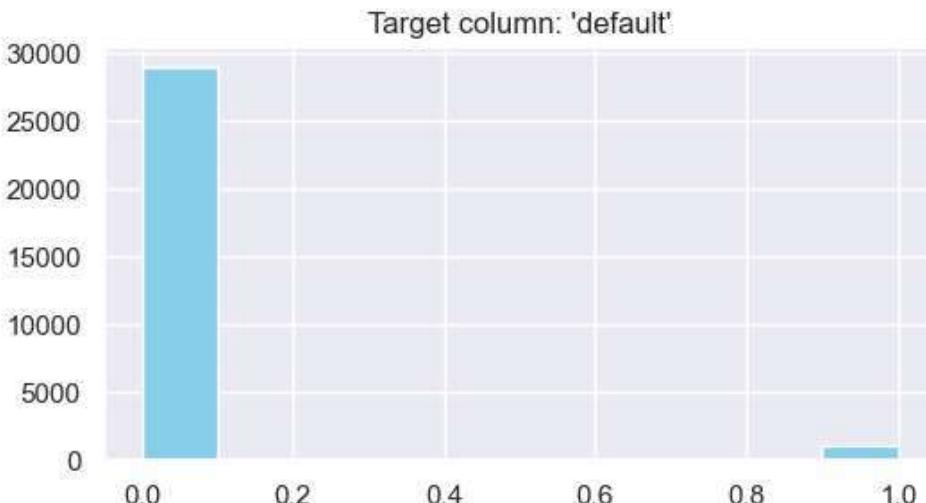
Out[11]:

(133, 60, 73, 1)

## Проверка на дисбаланс классов

Ввод [12]:

```
# Проверяем дисбаланс классов
plt.figure(figsize=(6,3))
train_nbki_df["default"].hist()
plt.title(f"Target column: 'default'")
plt.show()
pd.DataFrame(train_nbki_df["default"].value_counts())
```



Out[12]:

default	
0.0	29007
1.0	1000

### Дисбаланс

- В данных присутствует явный дисбаланс данных: 1 к 29. Это необходимо учесть перед обучением модели.

## Проверка на пропуски данных

Ввод [13]:

```
# Смотрим в каких столбцах есть пропуски данных
print("NaNs in train")
nan_contains = train_nbki_df.isna().sum()
display(pd.DataFrame(nan_contains[nan_contains > 0]))
print("NaNs in test")
nan_contains = test_nbki_df.isna().sum()
display(pd.DataFrame(nan_contains[nan_contains > 0]))
# В обучающей и в тестовых пропуски в одних и тех же признаках
```

NaNs in train

	0
6	1188
7	1089
9	69
10	198

NaNs in test

	0
6	1054
7	967
9	68
10	184

Ввод [14]:

```
# Проверяем, что пропуски данных это не особенность целевого таргета (банкротства)
nan_contains = train_nbki_df[train_nbki_df[target_column == 1].isna().sum()]
pd.DataFrame(nan_contains[nan_contains > 0])
```

Out[14]:

	0
6	29
7	22
10	3

Ввод [15]:

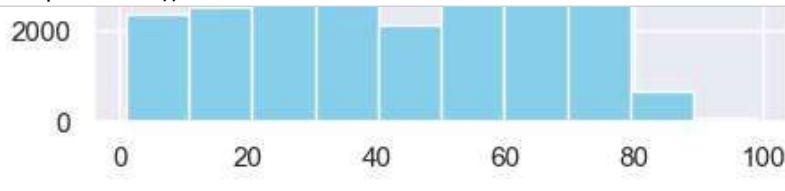
```
train_nbki_df[["6", "7", "9", "10"]].describe()
```

Out[15]:

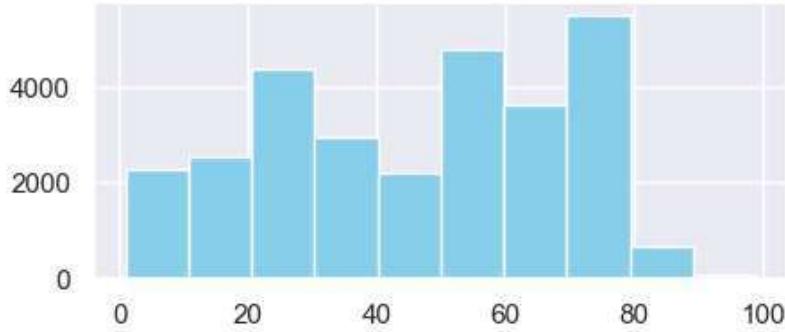
	6	7	9	10
<b>count</b>	28819.000000	28918.000000	29938.000000	29809.000000
<b>mean</b>	46.050036	45.760841	5.836562	0.555839
<b>std</b>	23.971219	23.747816	3.782330	0.496881
<b>min</b>	1.000000	1.000000	0.000000	0.000000
<b>25%</b>	24.000000	24.000000	2.000000	0.000000
<b>50%</b>	50.000000	50.000000	6.000000	1.000000
<b>75%</b>	66.000000	66.000000	9.000000	1.000000
<b>max</b>	99.000000	99.000000	13.000000	1.000000

Ввод [16]:

```
for c in ["6", "7", "9", "10"]:
    plt.figure(figsize=(5,2))
    train_nbki_df[c].hist()
    plt.title(f"column name: {c}")
    plt.show()
```



column name: 7



Ввод [17]:

```
def fillna_df(df):
    df["6"] = df["6"].fillna(df["6"].median())
    df["7"] = df["7"].fillna(df["7"].median())
    df["9"] = df["9"].fillna(df["9"].value_counts().sort_values(ascending=False).index[0])
    df["10"] = df["10"].fillna(df["10"].value_counts().sort_values(ascending=False).index[0])
    return df
```

Ввод [18]:

```
train_nbki_df = fillna_df(train_nbki_df)
test_nbki_df = fillna_df(test_nbki_df)
```

Ввод [19]:

```
# Проверяем, что не осталось значений NaN
nan_contains = train_nbki_df.isna().sum()
assert len(nan_contains[nan_contains > 0]) == 0, "Остались не учтенные NaN в обучающей выборке"
nan_contains = test_nbki_df.isna().sum()
assert len(nan_contains[nan_contains > 0]) == 0, "Остались не учтенные NaN в тестовой выборке"
```

## Пропуски в данных

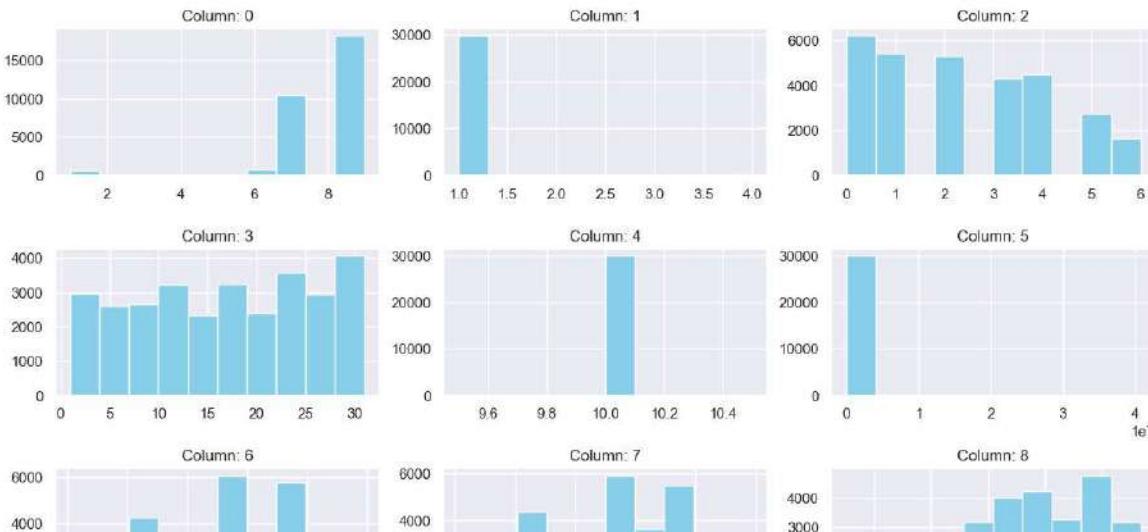
- Пропуски наблюдаются в колонках: 6,7,9,10
- В обучающей и в тестовых пропуски в одних и тех же признаках.
- Проверено, что пропуски данных - это не особенность именного целевого таргета (банкротства), можно произовдить заполнение пропусков по упращенному пути
- Распределение значений для "6", "7" находится между 0 и 100. Для заполнения пропусков будет использовано медианное значение
- Распределение значений для признака "9" находится между 0 и 12. Для заполнение пропусков будет использовано самое популярное значение
- Значения признака "10" - либо 0 либо 1. Для заполнения пропусков будет использовано самое популярное значение

## Распределений значений в признаках

Ввод [20]:

```
%time
# Визуализация распределений значений в данных
ncols = 3
nrows = len(feature_columns)//3 + 1
fig, axs = plt.subplots(nrows=nrows, ncols=3, figsize=(15, 3*nrows))
plt.subplots_adjust(hspace=0.5)
print("Распределение значений в обучающей выборке")
for column_name, ax in zip(feature_columns, axs.ravel()):
    train_nbki_df[column_name].hist(ax=ax)
    ax.set_title(f"Column: {column_name}")
plt.show()
```

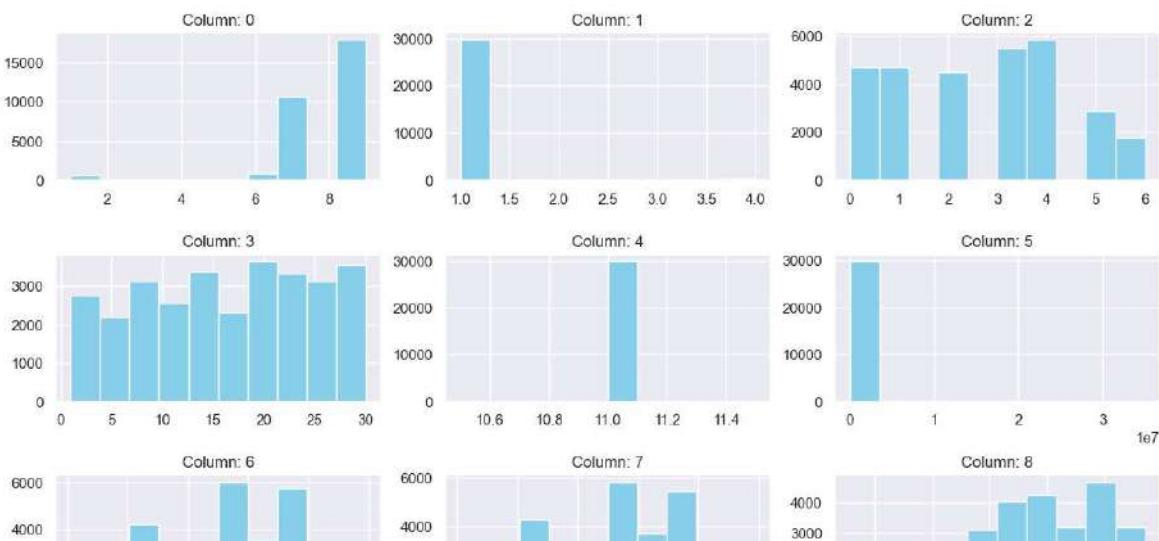
Распределение значений в обучающей выборке



Ввод [21]:

```
%%time
# Визуализация распределений значений в тестовых данных.
# !!!Тестовые данные трогать нельзя и по хорошему даже подсматривать за распределением не рекомендуется!
nrows = 3
ncols = len(feature_columns)//3 + 1
fig, axs = plt.subplots(nrows=nrows, ncols=cols, figsize=(15, 3*nrows))
plt.subplots_adjust(hspace=0.5)
print("Распределение значений в тестовой выборке")
for column_name, ax in zip(feature_columns, axs.ravel()):
    test_nbki_df[column_name].hist(ax=ax)
    ax.set_title(f"Column: {column_name}")
plt.show()
```

Распределение значений в тестовой выборке



Ввод [22]:

```
# Выявляем константные признаки
constant_features = []

# Базовый способ выявления констант, просто если одно значение - это константа
# for column in feature_columns:
#     if len(train_df[column].value_counts()) == 1:
#         constant_features.append(column)

# Продвинутый способ. За константу также принимаем те признаки у которых больше 99,9% значений в одном
for column in feature_columns:
    unique_labels, counts = np.unique(train_nbki_df[column].values, return_counts=True)
    proportions = counts / len(train_nbki_df[column].values)
    if max(proportions) > 0.999 and len(proportions) <= 2:
        print(f"column: {column}, imbalance > 0.999, proportions: {len(proportions)}, ({proportions[:2]})")
        constant_features.append(column)

# Исключаем из списка фичей, те фичи у которых значения константные
feature_columns = list(set(feature_columns) - set(constant_features))
categorical_columns = list(set(categorical_columns) - set(constant_features))
numeric_columns = list(set(numeric_columns) - set(constant_features))
len(feature_columns), len(categorical_columns), len(numeric_columns)

column: 4, imbalance > 0.999, proportions: 1, ([1.])
column: 58, imbalance > 0.999, proportions: 2, ([9.99966674e-01 3.33255574e-05])
column: 59, imbalance > 0.999, proportions: 2, ([9.99966674e-01 3.33255574e-05])
column: 62, imbalance > 0.999, proportions: 2, ([0.99900023 0.00099977])
column: 69, imbalance > 0.999, proportions: 1, ([1.])
column: 73, imbalance > 0.999, proportions: 1, ([1.])
```

Out[22]:

(127, 54, 73)

## Распределений значений

- Наблюдаются значения близкие к константным, например признак "1", у которого (значение "1" встречается 29768 раз, а значение "4" встречается 239 раз). Такие признаки необходимо исключить. Если только эти редкие значения не дают явный информационный выигрыш, например признак есть ли судимость, может встречается только у 1 из 10000 человек, но это является очень явным признаком ненадежности челоека. Варианты действий:
  - Удалить такого признаки, которые близки константным значемниям
  - Проверить корреляцию редких значений с искомым таргетом и тогда принять решение об удалении
- Наблюдается много признаков у которых основные значения находятся в одной области, но присутствуют аномальные значения (вероятно выбросы), из-за которых графики изображены в виде большого скопления значений в одном месте но ось абсцисс растянута, значит есть некоторое кол-во значений с аномальными значениями. Например признак "15". Варианты действий:
  - Можно оставить как есть, потому что для деревьев (планируется использовать бустинг) эти значения будут просто обработаны в рамках разделеющих правил деревьев. Также необходимо учитывать, что это могут быть не выбросы, а действительные просто такие данные.
  - Можно обрезать значения. Для значений больше какого-то порога, указывать значение этого порога
  - В случае использования линейных моделей (в том числе нейронных сетей) можно прологорифмировать эти признаки (предварительно решив что делать с нулевыми значениями), чтобы изменить масштаб и чтобы не возникали ненормальные веса из-за таких разбросов значений. В нашем же случае будет применятся бустинг (потому что он лучше работает с табличными данными, чем нейронки или линейные модели), поэтому можно не изменять масштаб данных

На **первом** этапе примем следующие решения:

- Константные признаки - удалим те признаки которые явно являются константами, остальные будут отсеяны на этапе построения матрицы корреляции
- Признаки с "выбросами" - на первом этапе оставить как есть

В дальнейшем возможно сделать пересмотр способа обработки указанных признаков

## Анализ значимости признаков

Анализ значимости признаков:

- корреляция
- p-value
- хи-квадрат (для категориальных признаков)

Ввод [23]:

```
%time  
# Формирование матрицы корреляции  
matrix_corr = train_nbki_df.corr()
```

Wall time: 1.35 s

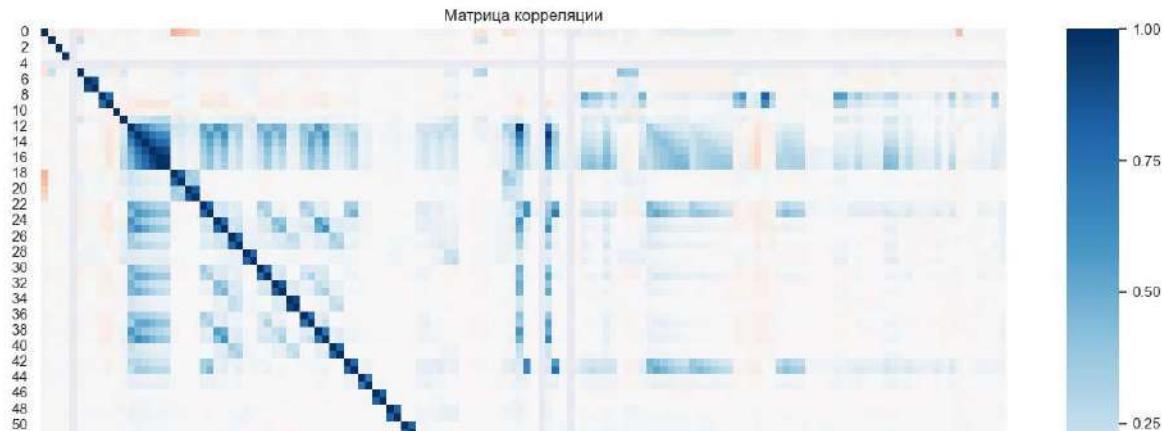
Ввод [24]:

```
%time  
# Вывод матрицы корреляции  
plt.figure(figsize=(16, 14))  
heatmap = sns.heatmap(matrix_corr, vmin=-1, vmax=1, cmap='RdBu')  
heatmap.set_title('Матрица корреляции');
```

Wall time: 409 ms

Out[24]:

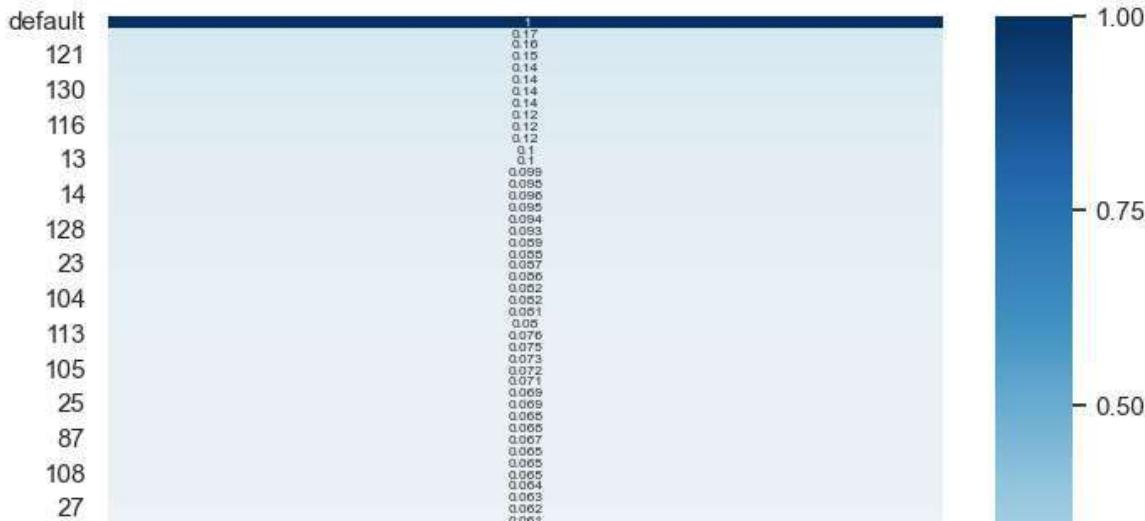
Text(0.5, 1.0, 'Матрица корреляции')



Ввод [25]:

```
# Признаки коррелирующие с банкротством
plt.figure(figsize=(8, 12))
heatmap = sns.heatmap(matrix_corr[[target_column]].sort_values(by=target_column, ascending=False), vmi=1)
heatmap.set_title('Корреляция признаков с target/дефолт', fontdict={'fontsize':18}, pad=16);
```

Корреляция признаков с target/дефолт



Ввод [26]:

```
abs(matrix_corr[[target_column]]).sort_values(by=target_column, ascending=False)[:10]
```

Out[26]:

default	
default	1.000000
131	0.165954
119	0.162577
121	0.146806
122	0.141112
123	0.139968
130	0.136896
120	0.135950
115	0.121643
116	0.121474

Ввод [31]:

```
# Топ признаков влияющие на факт банкротства:
matrix_corr[matrix_corr[target_column].abs() > 0.14][[target_column]]
```

Out[31]:

default	
119	0.162577
121	0.146806
122	0.141112
131	0.165954
default	1.000000

## Ввод [32]:

```
# Проверяем уровень значимости p-value и уровень корреляции с целевой функцией
high_p_value = []

for column in feature_columns:
    feat = train_nbki_df[column]
    target = train_nbki_df[target_column]
    res = stats.pearsonr(feat, target)
    corr, p_value = res
    if p_value > 0.05 and abs(corr) < 0.01:
        high_p_value.append(column)
        print(f"column: {column}, p_value ({p_value}) > 0.05, corr ({abs(corr)}) < 0.01")
```

```
column: 20, p_value (0.3028886806094353) > 0.05, corr (0.005947701484055303) < 0.01
column: 19, p_value (0.40449646638628595) > 0.05, corr (0.004812505057389107) < 0.01
column: 21, p_value (0.31843867318832797) > 0.05, corr (0.005759579419412774) < 0.01
column: 46, p_value (0.6885959806384963) > 0.05, corr (0.0023136189975489305) < 0.01
column: 32, p_value (0.11903532792395347) > 0.05, corr (0.008999083628844249) < 0.01
column: 80, p_value (0.45527982115414284) > 0.05, corr (0.004310395040919614) < 0.01
column: 118, p_value (0.1766388803072786) > 0.05, corr (0.007800317856476106) < 0.01
column: 110, p_value (0.1542809639427783) > 0.05, corr (0.008224014319956173) < 0.01
column: 47, p_value (0.11102202397158478) > 0.05, corr (0.009199862764263964) < 0.01
column: 7, p_value (0.5901072590276186) > 0.05, corr (0.0031098331718028204) < 0.01
column: 72, p_value (0.6804368046323256) > 0.05, corr (0.0023777343760227184) < 0.01
column: 68, p_value (0.7748438718869806) > 0.05, corr (0.0016513549756145022) < 0.01
column: 18, p_value (0.5217478079966877) > 0.05, corr (0.0036985248906044594) < 0.01
column: 50, p_value (0.3288796453082263) > 0.05, corr (0.005636624766577136) < 0.01
column: 2, p_value (0.12420317080947886) > 0.05, corr (0.008875158995757531) < 0.01
column: 97, p_value (0.6705670925127187) > 0.05, corr (0.0024556860131153603) < 0.01
column: 79, p_value (0.5221587223328767) > 0.05, corr (0.0036948752431601338) < 0.01
column: 99, p_value (0.7407846816113596) > 0.05, corr (0.0019098128682914295) < 0.01
column: 101, p_value (0.2254338713175685) > 0.05, corr (0.006998059012259234) < 0.01
```

## Ввод [29]:

```
%time
from sklearn.feature_selection import chi2
chi2_res = chi2(train_nbki_df[categorical_columns], pd.DataFrame(target))
chi2_threshold = np.quantile(chi2_res, 0.5) # порог хи-квадрат будем рассчитывать по заданному квантили
for i, cat_column in enumerate(categorical_columns):
    chi2_stat = chi2_res[0][i]
    p_value = chi2_res[1][i]
    if chi2_stat > chi2_threshold and p_value > 0.05:
        high_p_value.append(cat_column)
        print(f"column: {cat_column}, p_value ({p_value}) > 0.05, chi2_stat ({chi2_stat}) -> high value")
```

```
column: 20, p_value (0.11721405254970706) > 0.05, chi2_stat (2.4541574000211286) -> high value
column: 50, p_value (0.2439600991933647) > 0.05, chi2_stat (1.3575640516705798) -> high value
column: 47, p_value (0.07232295138355575) > 0.05, chi2_stat (3.2294958227419333) -> high value
column: 51, p_value (0.055612968947366) > 0.05, chi2_stat (3.663611324649832) -> high value
column: 19, p_value (0.10677237727176309) > 0.05, chi2_stat (2.601355230972631) -> high value
column: 21, p_value (0.12691228422328535) > 0.05, chi2_stat (2.329872348494016) -> high value
column: 2, p_value (0.06687333606883193) > 0.05, chi2_stat (3.358145524819537) -> high value
column: 18, p_value (0.23739622273827818) > 0.05, chi2_stat (1.3959894529783654) -> high value
Wall time: 953 ms
```

Ввод [30]:

```
# Оценка Z-score
# В Z-score предполагается, что каждое наблюдение, для которого значение статистики больше 3 или меньше
# Т.е. на сколько стандартных отклонений (сигм) значения признака меньше или больше генеральной совокупности
z_score = np.abs(stats.zscore(train_nbki_df[feature_columns]))
train_nbki_df[feature_columns][(z_score < 3).all(axis=1)].shape, train_nbki_df.shape
# Примерно половина объектов имеет признаки у которых отклонение от генеральной выборки отличается на
# Мы не будем их исключать, потому что деревья (в частности бустинг) сам распределит такие выбросы по деревьям
# Данную информацию просто принимаем к сведению и при необходимости дальнейшем сделаем более детальный разбор
```

Out[30]:

(14178, 127), (30007, 134)

Ввод [31]:

```
# На данном этапе не производим преднамеренное исключение слабокоррелирующих признаков
# Исключаем из списка фичей, те фичи у которых значения константные
# feature_columns = list(set(feature_columns) - set(high_p_value))
# categorical_columns = list(set(categorical_columns) - set(high_p_value))
# numeric_columns = list(set(numeric_columns) - set(high_p_value))
# len(feature_columns), len(categorical_columns), len(numeric_columns)
```

## Значимость признаков

- По матрице корреляции наблюдается ряд признаков сильно коррелирующих между собой. Данные признаки надо будет фильтровать на этапе построения модели и на этапе генерации новых признаков
- Проверяем p-value и коэффициент корреляции Пирсона между каждым числовым признаком и целевой переменой. Чем ниже p-value, тем больше статистическая значимость рассматриваемых признаков. Значение p-value меньшее или равное 0,05 считается статистически значимым. Признаки с большим значением p-value (признак стат. не значим) и с маленькой корреляцией (связь признака с таргетом не обнаружена) стоит исключить из рассмотрения, но на данном этапе не производим исключений слабокоррелирующих признаков. В дальнейшем будет произведен дополнительный анализ.
- Для категориальных признаков также проверяется стат-значимость через Хи-квадрат. В дальнейшем для ужесточения отбора признаков можно повысить порог стат-значимости
- Примерно половина объектов имеет признаки у которых отклонение от генеральной выборки отличается на более чем 3 сигмы. Мы не будем их исключать, потому что деревья (в частности бустинг) сам распределит такие выбросы по нужным веткам (если будут выбраны соответствующие решающие правила). Данную информацию просто принимаем к сведению и при необходимости дальнейшем сделаем более детальный разбор для каждого признака

## Формирование train/val/test

Деление данных на train/val/test надо делать с учетом дисбаланса классов

Ввод [33]:

```
train_df, val_df = train_test_split(train_nbki_df[feature_columns + [target_column]], test_size=0.2, stratify=train_nbki_df[target_column])
val_df, test_df = train_test_split(val_df, test_size=0.5, shuffle=True, random_state=53, stratify=val_df[target_column])
test_true_df = test_nbki_df[feature_columns + [target_column]].copy()
```

Ввод [34]:

```
# Учитывая, что в таргете сильный дисбаланс, тогда проверяем, что распределение val/train равно исходному
v_cnt_nbki_df = train_nbki_df[target_column].value_counts()
v_cnt_train = train_df[target_column].value_counts()
v_cnt_val = val_df[target_column].value_counts()
v_cnt_test = test_df[target_column].value_counts()
assert round(v_cnt_nbki_df[1]/v_cnt_nbki_df[0], 4) == round(v_cnt_train[1]/v_cnt_train[0], 4)
assert round(v_cnt_nbki_df[1]/v_cnt_nbki_df[0], 4) == round(v_cnt_val[1]/v_cnt_val[0], 4)
assert round(v_cnt_nbki_df[1]/v_cnt_nbki_df[0], 4) == round(v_cnt_test[1]/v_cnt_test[0], 4)
```

Ввод [35]:

```
X_train = train_df[feature_columns]
y_train = train_df[target_column]
X_val = val_df[feature_columns]
y_val = val_df[target_column]
X_test = test_df[feature_columns]
y_test = test_df[target_column]

X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test.shape, y_test.shape
```

Out[35]:

((24005, 127), (24005,), (3001, 127), (3001,), (3001, 127), (3001,))

## Построение модели

### Baseline Модель

Ввод [36]:

```
# Вывод графика feature importance
def plot_feature_importance(importance, names, model_name="", top_n=-1):
    """Функция вывода feature importance
    :importance - массив важности фичей, полученный от модели
    :names - массив названий фичей
    :model_name - название модели
    :top_n - кол-во выводимых фичей

    :return - fi_df - feature importance датафрейм
    """
    feature_importance = np.array(importance)
    feature_names = np.array(names)

    data={'feature_names':feature_names,'feature_importance':feature_importance}
    fi_df = pd.DataFrame(data)
    fi_df.sort_values(by=['feature_importance'], ascending=False,inplace=True)

    plt.figure(figsize=(10,8))
    sns.barplot(x=fi_df['feature_importance'][:top_n], y=fi_df['feature_names'][:top_n])
    if top_n != -1:
        plt.title(f"{model_name} FEATURE IMPORTANCE (Top: {top_n})")
    else:
        plt.title(f"{model_name} FEATURE IMPORTANCE")
    plt.xlabel('FEATURE IMPORTANCE')
    plt.ylabel('FEATURE NAMES')
    return fi_df
```

Ввод [37]:

```
# Вывод графика ROC-AUC
def plot_roc_auc(y_true, y_pred):
    fpr, tpr, _ = roc_curve(y_true=y_true, y_score=y_pred)
    roc_auc = roc_auc_score(y_true=y_true, y_score=y_pred)

    plt.figure(figsize=(10, 3))
    plt.plot(fpr, tpr, color='darkorange',
              lw=2, label='ROC curve (area = %0.4f)' % roc_auc, alpha=0.5)

    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', alpha=0.5)

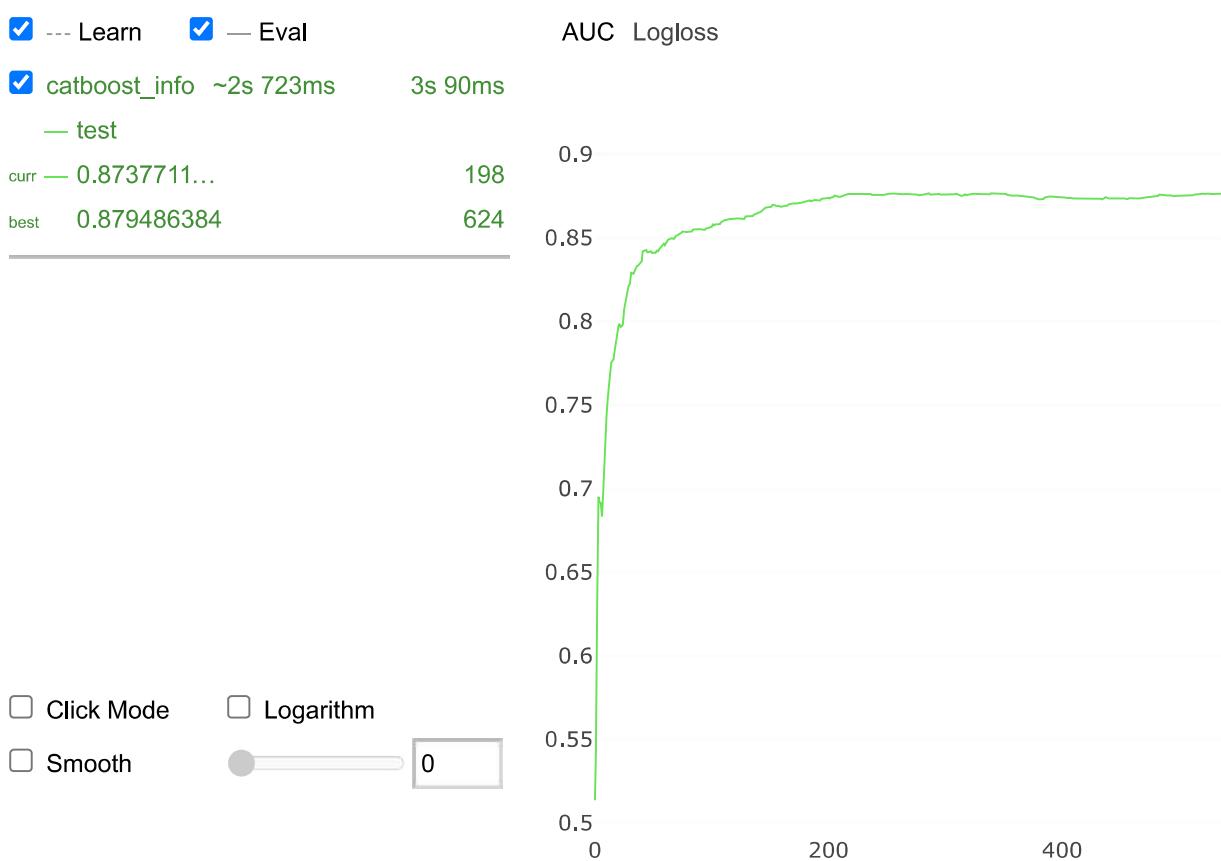
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)
    plt.grid(True)
    plt.xlabel('False Positive Rate', fontsize=12)
    plt.ylabel('True Positive Rate', fontsize=12)
    plt.title('Receiver operating characteristic', fontsize=16)
    plt.legend(loc="lower right", fontsize=12)
    plt.show()
    return roc_auc
```

Ввод [38]:

```
def calc_gini(auc_roc):
    gini = 2 * auc_roc - 1
    return gini
```

Ввод [54]:

```
# baseline Модель
default_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=200, random_state=53)
default_model.fit(X_train, y_train, eval_set=(X_val, y_val), plot=True, verbose=False)
```

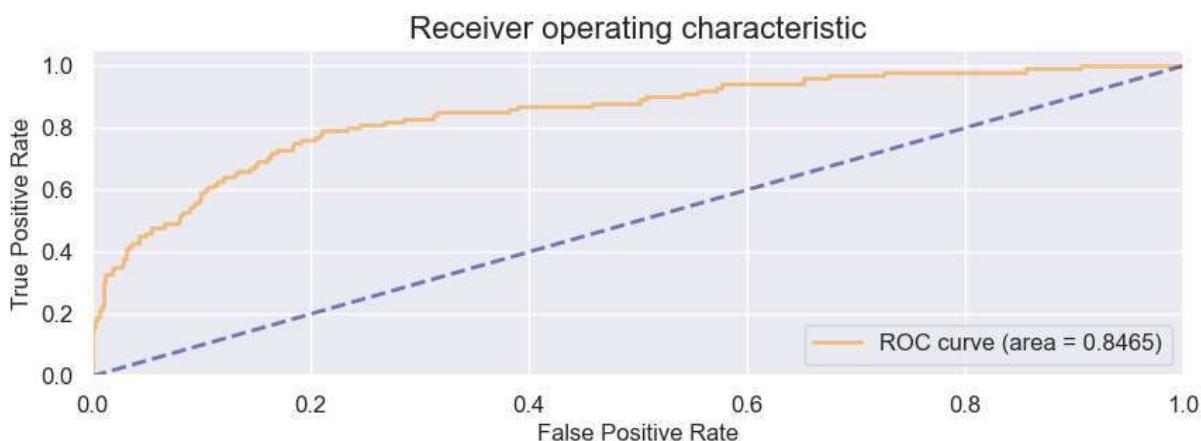


Out[54]:

```
<catboost.core.CatBoostClassifier at 0x211725a2188>
```

Ввод [55]:

```
# Делаем предсказание на тестовых данных
y_pred_default = default_model.predict_proba(X_test)[:,1]
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test, y_pred=y_pred_default)
print(f"ROC-AUC: {roc_auc}, Gini: {calc_gini(roc_auc)}")
```



ROC-AUC: 0.8464701826956222, Gini: 0.6929403653912445

Ввод [56]:

```
# Переводим вероятностные предсказания в класс
threshold = 0.5
y_pred_default_class = (y_pred_default > threshold).astype(int)
```

Ввод [57]:

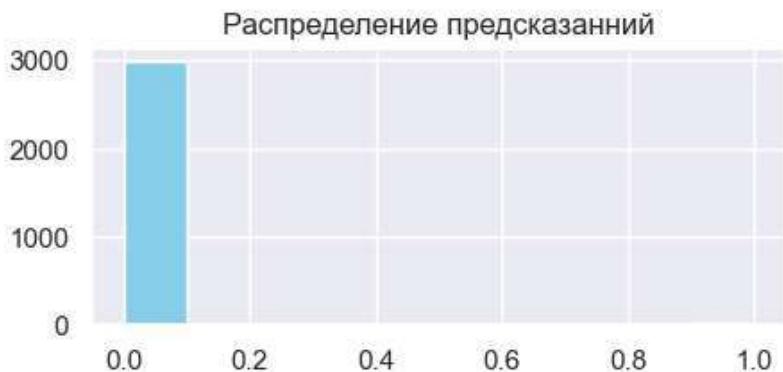
```
print(f"Распределение исходного таргета: {sum(y_test)/len(y_test)}")
print(f"Распределение предсказанного таргета: {sum(y_pred_default_class)/len(y_pred_default_class)}")

fig = plt.figure(figsize=(5,2))
pd.DataFrame(y_test).hist(ax=fig.gca())
plt.title("Распределение исходных данных")

fig = plt.figure(figsize=(5,2))
pd.DataFrame(y_pred_default_class).hist(ax=fig.gca())
plt.title("Распределение предсказанный")

plt.show()
```

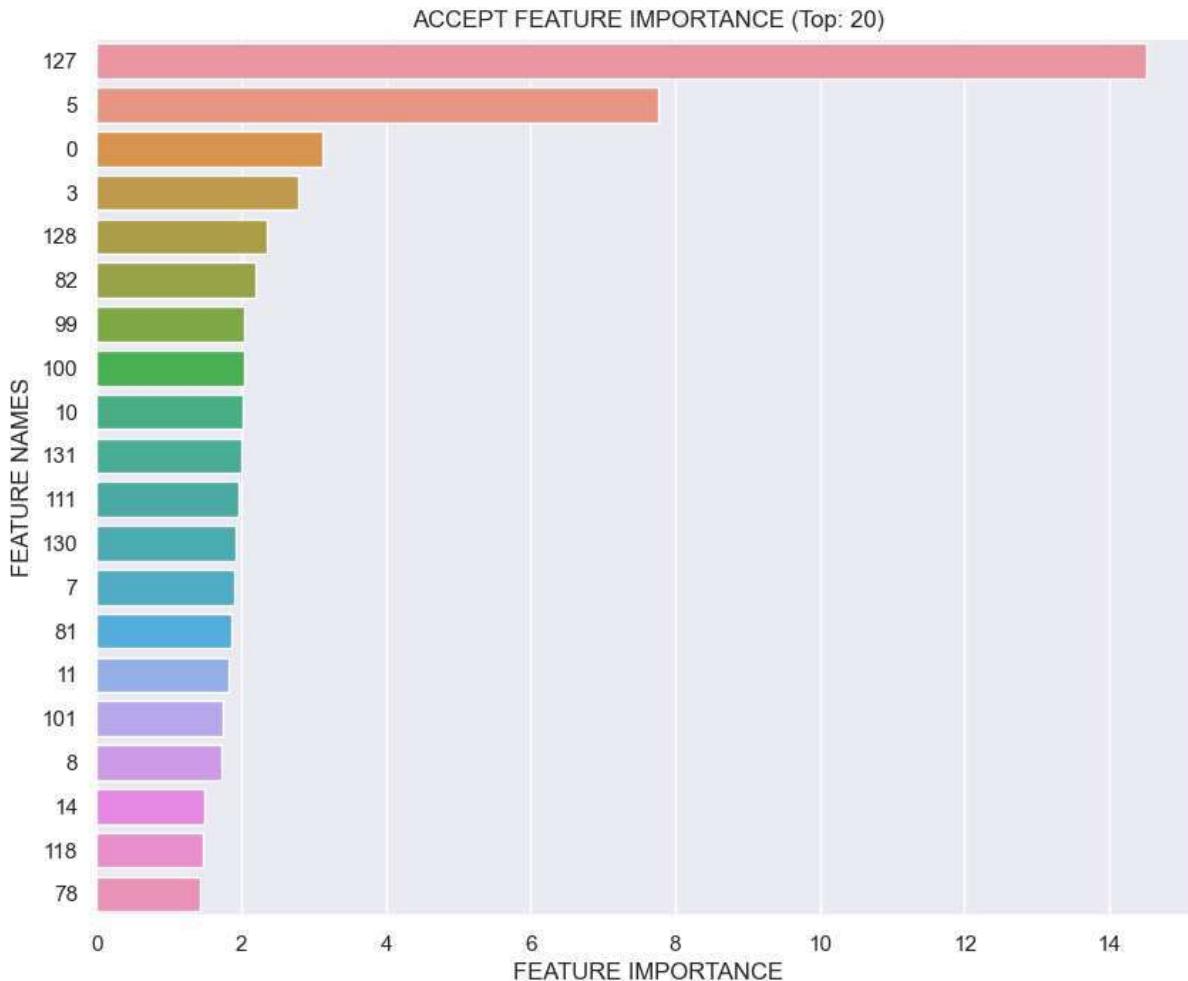
Распределение исходного таргета: 0.03332222592469177  
Распределение предсказанного таргета: 0.005664778407197601



Ввод [59]:

# Важность признаков

fi\_df = plot\_feature\_importance(default\_model.get\_feature\_importance(), X\_test.columns, model\_name='ACI')

**Промежуточный итог:**

- Выявлено сильное влияние параметра "127" на предсказания модели.
- Доля дефолтов в исходных данных: 0.0333, а в предсказанных: 0.0056. Это результат большого дисбаланса классов
- Следующий шаг борьба с дисбалансом классов

**Варианты улучшения:**

- Борьба с дисбалансом
- Выбор порога threshold
- Feature engineering
- Подбор гиперпараметров
- Увеличение данных для обучения. Использование валидационных данных за счет кросс-валидации

**Результаты экспериментов:**

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Gini на test	Доля дефолтов в исходных данных	Доля предсказанных дефолтов
1	Базовая модель	Простая модель	0.8794	0.8464	0.6929	0.0333	0.0056

## Добавляем балансировку классов

Для учета дисбаланса классов используем compute\_class\_weight от sklearn

Ввод [60]:

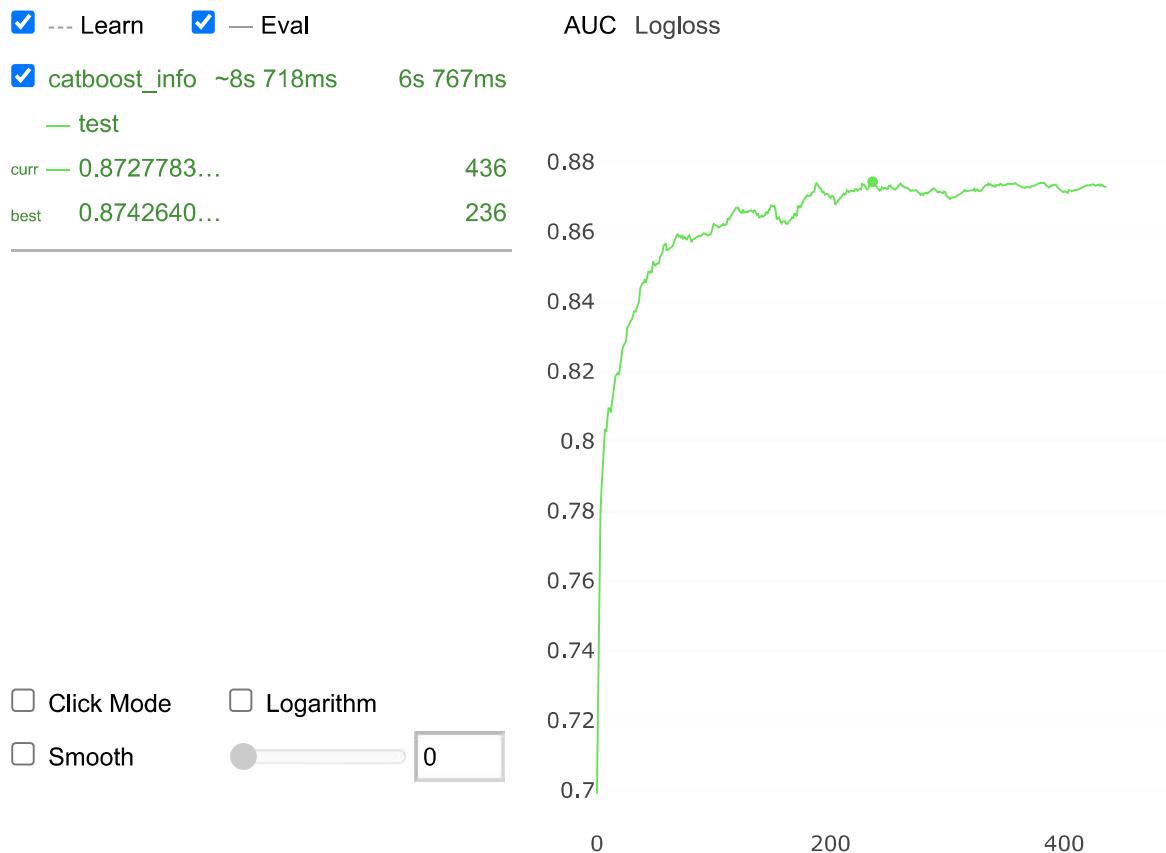
```
# Расчет дисбаланса классов
classes = np.unique(y_train)
weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train)
class_weights = dict(zip(classes, weights))
class_weights
```

Out[60]:

```
{0.0: 0.5172376642964879, 1.0: 15.003125}
```

Ввод [61]:

```
# Обучаем модель модель
default_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=200, random_state=53, cl
default_model.fit(X_train, y_train, eval_set=(X_val, y_val), plot=True, verbose=False)
```

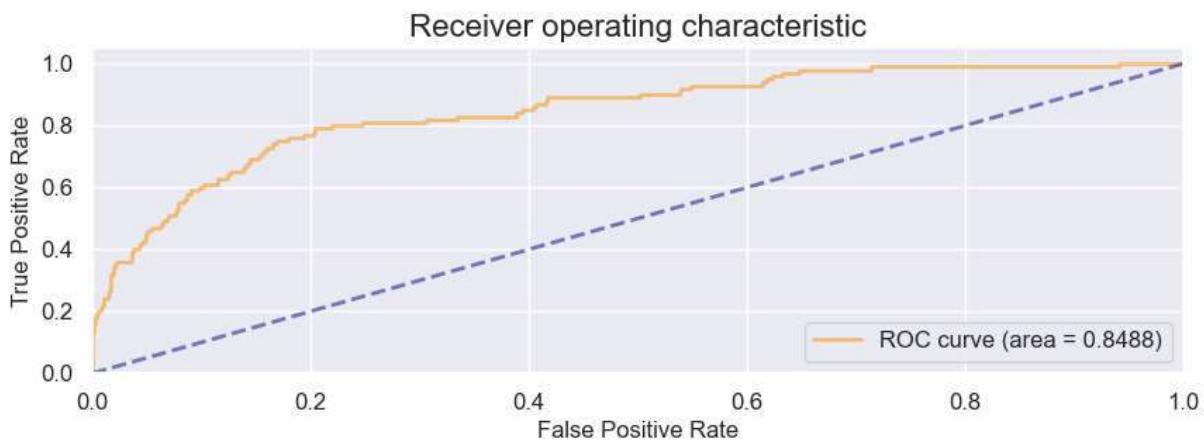


Out[61]:

```
<catboost.core.CatBoostClassifier at 0x2116cd30388>
```

Ввод [62]:

```
# Делаем предсказание на тестовых данных
y_pred_default = default_model.predict_proba(X_test)[:,1]
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test, y_pred=y_pred_default)
print(f"ROC-AUC: {roc_auc}, Gini: {calc_gini(roc_auc)}")
```



ROC-AUC: 0.8488279903481559, Gini: 0.6976559806963119

Ввод [63]:

```
# Переводим вероятностные предсказания в класс
threshold = 0.5
y_pred_default_class = (y_pred_default > threshold).astype(int)
```

## Ввод [64]:

```

print(f"Распределение исходного таргета: {sum(y_test)/len(y_test)}")
print(f"Распределение предсказанного таргета: {sum(y_pred_default_class)/len(y_pred_default_class)}")

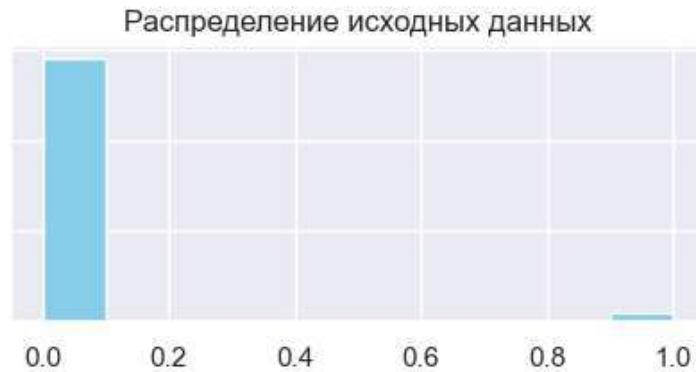
fig = plt.figure(figsize=(5,2))
pd.DataFrame(y_test).hist(ax=fig.gca())
plt.title("Распределение исходных данных")

fig = plt.figure(figsize=(5,2))
pd.DataFrame(y_pred_default_class).hist(ax=fig.gca())
plt.title("Распределение предсказанный")

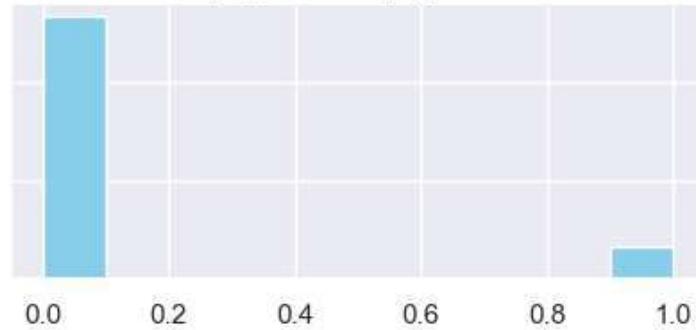
plt.show()

```

Распределение исходного таргета: 0.03332222592469177  
 Распределение предсказанного таргета: 0.10596467844051982



Распределение предсказаний



## Промежуточный итог:

- Балансировка улучшила метрику
- Теперь модель старается предсказать редкий класс, распределение предсказаний больше похоже на исходное распределение

## Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Gini на test	Доля дефолтов в исходных данных	Доля предсказанных дефолтов
1	Базовая модель	Простая модель	0.8794	0.8464	0.6929	0.0333	0.0056
2	Модель №1 + балансировка классов	Учитываем в модели дисбаланс классов	0.8742	0.8488	0.6976	0.0333	0.1059

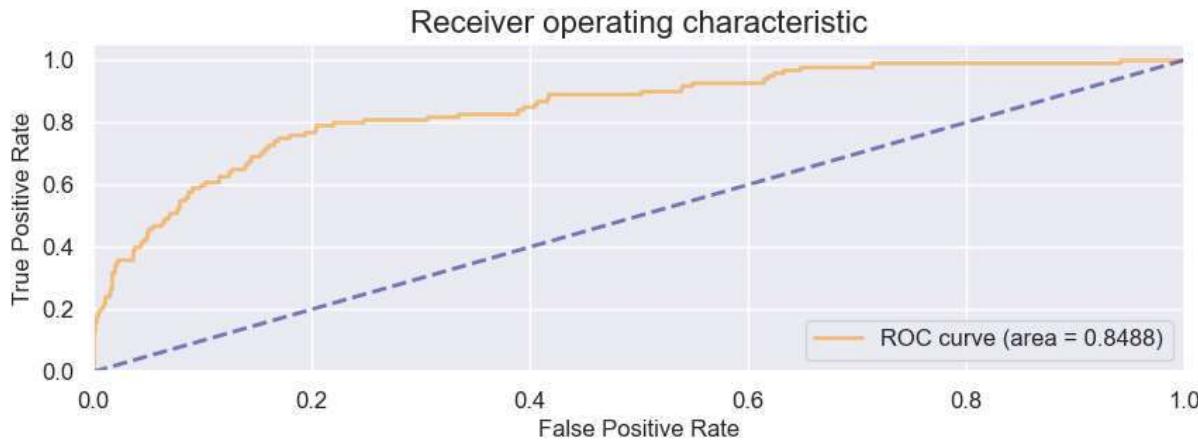
## Выбор порога threshold

Ввод [66]:

```
# Делаем предсказание модели (на тестовых данных) через predict_proba
y_pred_proba_default = default_model.predict_proba(X_test)
```

Ввод [68]:

```
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test, y_pred=y_pred_default)
print(f"ROC-AUC: {roc_auc}, Gini: {calc_gini(roc_auc)}")
```



ROC-AUC: 0.8488279903481559, Gini: 0.6976559806963119

Ввод [69]:

```
# Определяем оптимальный порог как поиск максимального значения для разности tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=y_pred_proba_default[:, 1])
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")
```

Оптимальный порог: 0.33018554682788337

Ввод [71]:

```
# Переводим вероятностные предсказания в класс
y_pred_default_class = (y_pred_default > best_thresh).astype(int)
```

Ввод [72]:

```
print(f"Распределение исходного таргета: {sum(y_test)/len(y_test)}")
print(f"Распределение предсказанного таргета: {sum(y_pred_default_class)/len(y_pred_default_class)}")
```

Распределение исходного таргета: 0.03332222592469177  
 Распределение предсказанного таргета: 0.22359213595468178

### Промежуточный итог:

- Выбор порога (threshold) улучшил итоговую метрику AUC
- Теперь модель старается предсказать редкий класс, распределение предсказаний больше похоже на исходное распределение

### Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Gini на test	Доля предсказанных дефолтов
1	Базовая модель	Простая модель	0.8794	0.8464	0.6929	0.0056

2	Модель №1 + балансировка классов	Учитываем в модели дисбаланс классов	0.8742	0.8488	0.6976	0.1059
3	Модель №2 + оптимальный threshold	Определяем оптимальный порог (threshold) для принятия решения о банкротстве. Модель не изменилась	0.8742	0.8488	0.6976	0.2235

## Выделяем категориальные признаки

На данный момент все признаки в модель подаются как числовые, это значит что если в данных есть категориальные признаки (которые закодированы 1, 2, 3, 4), то они используют по сути label-кодирование, когда категории дается порядковый номер. Это не лучший способ кодирования категориальных признаков, гораздо лучше ONE или target-кодирование. Данная гипотеза требует проверки и не факт, что покажет хорошие результаты.

Можно самостоятельно сделать кодирование категориальных признаков, но в catboost под капотом как раз используется, что-то похожее на target-кодирование. Для catboost достаточно указать какие признаки являются категориальными, необходимое кодирование будет выполнено под капотом. Поэтому для быстрой проверки данного предположения воспользуемся как раз средствами catboost. Для того чтобы числа перевести в категории достаточно перевести значения из числовых в текстовые и сообщить catboost список категориальных признаков

Ввод [73]:

```
threshold_cardinality = 20
categorical_columns = list(set(check_cardinality[(check_cardinality <= threshold_cardinality)].index))
categorical_columns = list(set(feature_columns)&set(categorical_columns))
print(f"Кол-во признаков попадающих под тип категориальный: {len(categorical_columns)}")
```

Кол-во признаков попадающих под тип категориальный: 54

Ввод [74]:

```
X_train_category = X_train.copy()
X_val_category = X_val.copy()
X_test_category = X_test.copy()

X_train_category[categorical_columns] = X_train_category[categorical_columns].astype(str)
X_val_category[categorical_columns] = X_val_category[categorical_columns].astype(str)
X_test_category[categorical_columns] = X_test_category[categorical_columns].astype(str)
```

Ввод [75]:

# Обучаем модель модель

```
default_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=200, random_state=53, class_weights=[1, 1])
default_model.fit(X_train_category, y_train, eval_set=(X_val_category, y_val), plot=True, verbose=False)
```

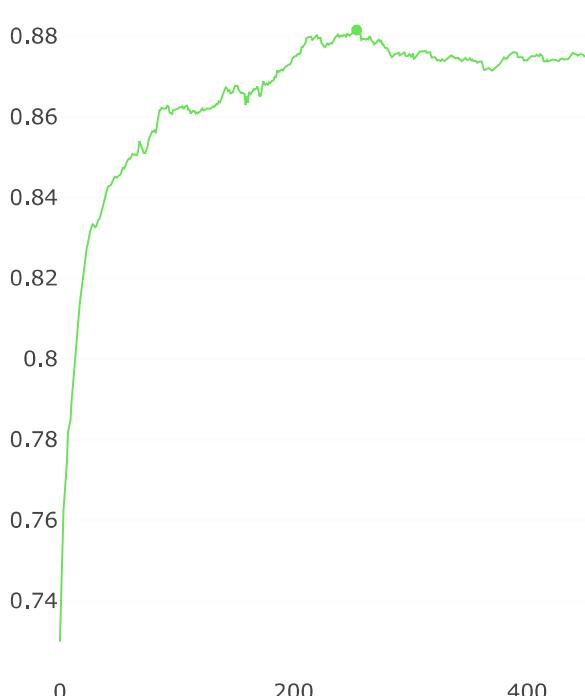
 --- Learn     — Eval

AUC Logloss

 catboost\_info ~1m 39s    15s 957ms test

curr 0.8620234...    88

best 0.8814960...    254

 Click Mode Logarithm Smooth Smooth

0

out[75]:

&lt;catboost.core.CatBoostClassifier at 0x2116ddd8448&gt;

## Ввод [78]:

```
# Делаем предсказание на тестовых данных
y_pred_default = default_model.predict_proba(X_test_category)[:,1]

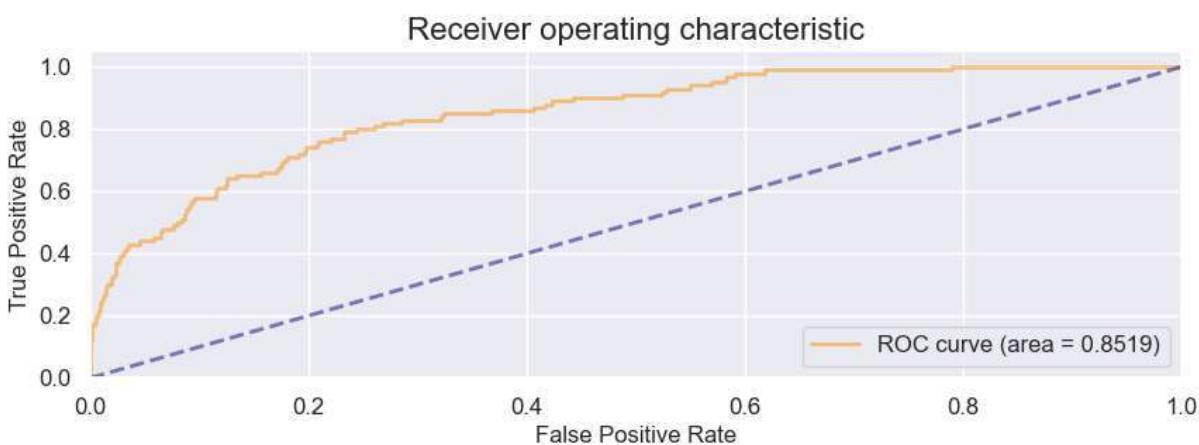
# Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=y_pred_default)
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")

# Формируем итоговое предсказание с учетом выбранного порога
y_pred_default_class = (y_pred_default > best_thresh).astype(int)

# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test, y_pred=y_pred_default)
print(f"ROC-AUC: {roc_auc}, Gini: {calc_gini(roc_auc)}")

print(f"Распределение исходного таргета: {sum(y_test)/len(y_test)}")
print(f"Распределение предсказанного таргета: {sum(y_pred_default_class)/len(y_pred_default_class)}")
```

Оптимальный порог: 0.3261619368609995



ROC-AUC: 0.8518855567045847, Gini: 0.7037711134091693  
 Распределение исходного таргета: 0.03332222592469177  
 Распределение предсказанного таргета: 0.25091636121292904

## Промежуточный итог:

- Гипотеза: если в данных есть категориальные признаки (которые закодированы 1, 2, 3, 4), то они по сути представлены в виде label-кодирования (когда категории дается порядковый номер). Это не лучший способ кодирования категориальных признаков, гораздо лучше ОНЕ или target-кодирование.
- Выделение ряда признаки в категориальный дало прирост ROC-AUC с 0.8488 до 0.8518. Можно считать, что гипотеза подтвердилась.
- По факту признаки с малым кол-ом уникальных значений переведены в тип категориальные

## Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Gini на test	Доля предсказанных дефолтов
1	Базовая модель	Простая модель	0.8794	0.8464	0.6929	0.0056
2	Модель №1 + балансировка классов	Учитываем в модели дисбаланс классов	0.8742	0.8488	0.6976	0.1059
3	Модель №2 + оптимальный threshold	Определяем оптимальный порог (threshold) для принятия решения о банкротстве. Модель не изменилась	0.8742	0.8488	0.6976	0.2235

## Генерация новых признаков (Feature engineering)

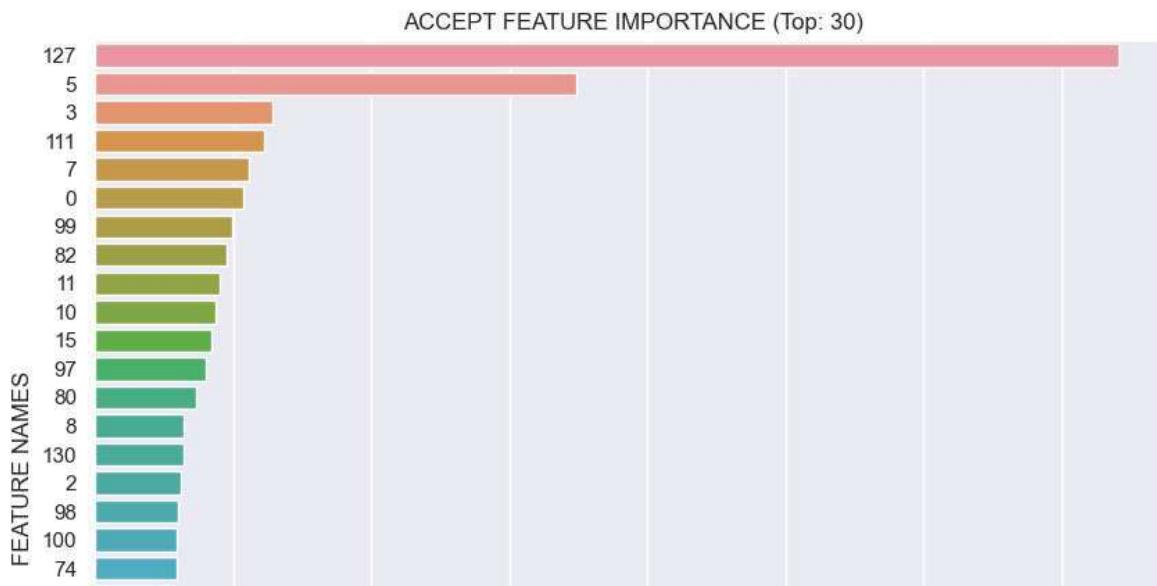
Генерация признаков планируется по следующему плану:

- Выбор топ N признаков которые имеют наибольшее влияние на модель - feature importance (используется предыдуще построенная модель)
- Выбор топ N признаков имеющих наибольшую корреляцию с таргетом
- Генерация отношений и произведений между выбранными признаками
- Обучение и оценка модели с новыми признаками

Ввод [79]:

```
# Посмотрим на Топ 30 признаков, которые имеют влияние на модель
```

```
fi_df = plot_feature_importance(default_model.get_feature_importance(), X_test.columns, model_name='AC...
```



Ввод [80]:

```
train_nbki_df_generated = train_nbki_df[feature_columns + [target_column]].copy()
train_nbki_df_generated.shape
```

Out[80]:

(30007, 128)

Ввод [81]:

```
%%time
# Формируем матрицу корреляции
corr_matrix = train_nbki_df_generated.corr()
corr_matrix.shape
```

Wall time: 1.23 s

Out[81]:

(128, 128)

Ввод [82]:

```
%time
n = 10
# Берем N признаков с наибольшей корреляцией с таргетом
base_features = set(abs(corr_matrix[target_column].drop([target_column], axis=0)).sort_values(ascending=False).head(n))
# Берем N признаков с наиболее важных для модели и объединяем с лучшими скоррелированными
base_features |= set(fi_df.sort_values(by=["feature_importance"], ascending=False)[:n]["feature_names"])
base_features = base_features - set(categorical_columns)
base_features = list(base_features)
print(f"base_features: {len(base_features)}")

e = 1e-3 # добавляем минимальное значение, чтобы уйти от деления на ноль и появления бесконечностей
# Сгенерируем новые признаки, которые будут представлять: отношение одного к другому и произведение обоих
for i in range(len(base_features)):
    base_f_1 = base_features[i]
    for j in range(i+1, len(base_features)):
        base_f_2 = base_features[j]
        train_nbki_df_generated[f"new_rlt_{base_f_1}_{base_f_2}"] = (train_nbki_df_generated[base_f_1] - e) / (train_nbki_df_generated[base_f_2] - e)
        train_nbki_df_generated[f"new_mltp_{base_f_1}_{base_f_2}"] = train_nbki_df_generated[base_f_1] * train_nbki_df_generated[base_f_2]
train_nbki_df_generated.shape
```

```
base_features: 15
Wall time: 192 ms
```

```
D:\_Work\_Projects\_Conda\DL52\lib\site-packages\ipykernel_launcher.py:17: PerformanceWarning: DataFrame is highly fragmented. This is usually the result of calling `frame.insert` many times, which has poor performance. Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()``app.launch_new_instance()
D:\_Work\_Projects\_Conda\DL52\lib\site-packages\ipykernel_launcher.py:18: PerformanceWarning: DataFrame is highly fragmented. This is usually the result of calling `frame.insert` many times, which has poor performance. Consider joining all columns at once using pd.concat(axis=1) instead. To get a de-fragmented frame, use `newframe = frame.copy()`
```

Out[82]:

(30007, 338)

Ввод [83]:

```
gen_feature_columns = list(train_nbki_df_generated.columns.drop([target_column]))
len(gen_feature_columns)
```

Out[83]:

337

Ввод [84]:

```
train_df, val_df = train_test_split(train_nbki_df_generated[gen_feature_columns + [target_column]], test_size=0.2, random_state=42)
val_df, test_df = train_test_split(val_df, test_size=0.5, shuffle=True, random_state=53, stratify=val_df[target_column])
X_train = train_df[gen_feature_columns]
y_train = train_df[target_column]
X_val = val_df[gen_feature_columns]
y_val = val_df[target_column]
X_test = test_df[gen_feature_columns]
y_test = test_df[target_column]
```

Ввод [85]:

```
X_train_category = X_train.copy()  
X_val_category = X_val.copy()  
X_test_category = X_test.copy()  
  
X_train_category[categorical_columns] = X_train_category[categorical_columns].astype(str)  
X_val_category[categorical_columns] = X_val_category[categorical_columns].astype(str)  
X_test_category[categorical_columns] = X_test_category[categorical_columns].astype(str)
```

Ввод [86]:

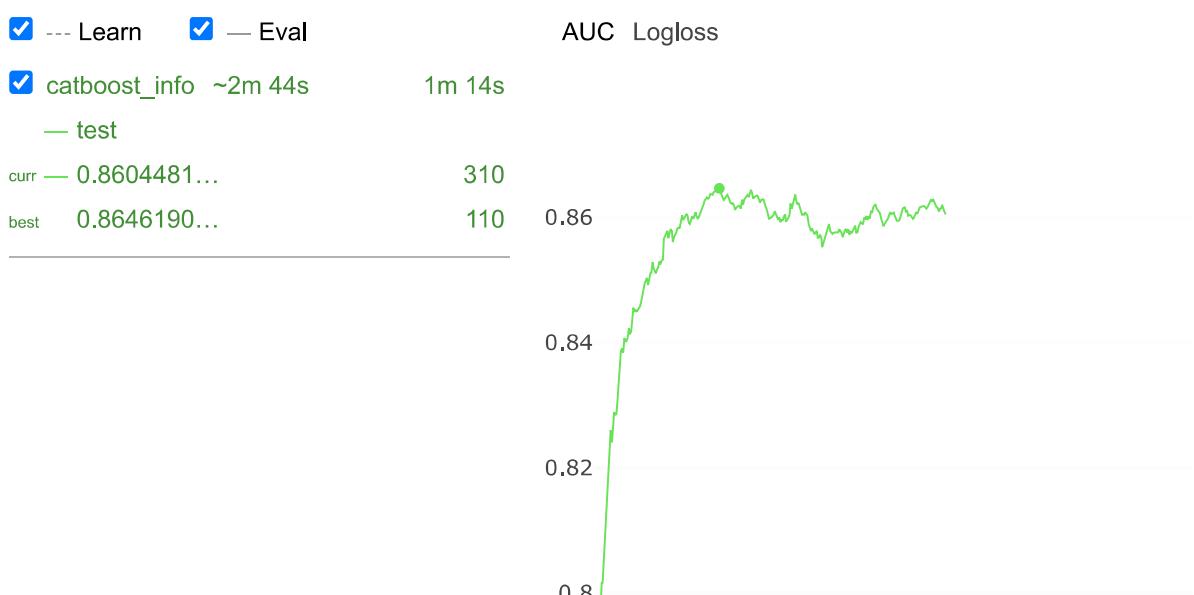
```
X_train_category.shape
```

Out[86]:

```
(24005, 337)
```

Ввод [87]:

```
# Обучаем модель модель  
default_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=200, random_state=53, class_weights="Balanced")  
default_model.fit(X_train_category, y_train, eval_set=(X_val_category, y_val), plot=True, verbose=False)
```



Ввод [88]:

```
# Делаем предсказание на тестовых данных
y_pred_default = default_model.predict_proba(X_test_category)[:,1]

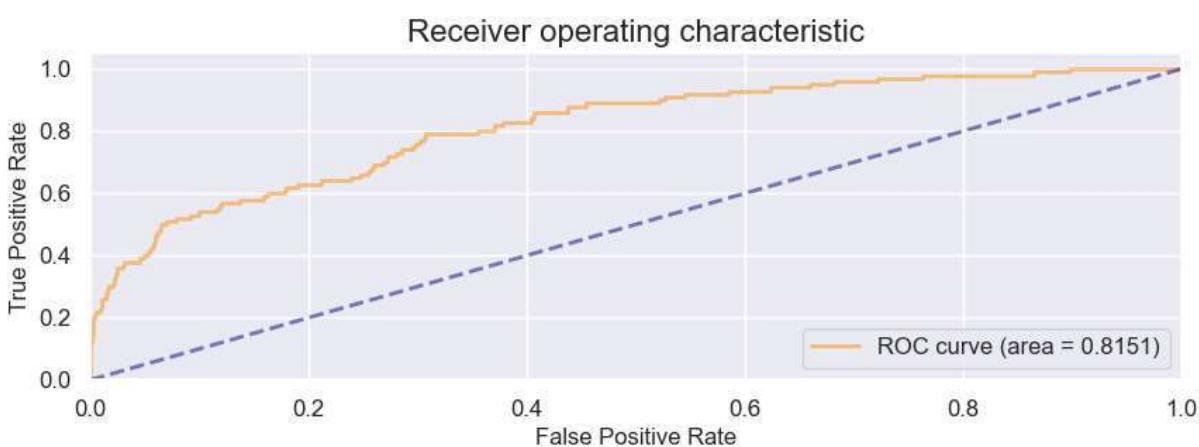
# Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=y_pred_default)
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")

# Формируем итоговое предсказание с учетом выбранного порога
y_pred_default_class = (y_pred_default > best_thresh).astype(int)

# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test, y_pred=y_pred_default)
print(f"ROC-AUC: {roc_auc}, Gini: {calc_gini(roc_auc)}")

print(f"Распределение исходного таргета: {sum(y_test)/len(y_test)}")
print(f"Распределение предсказанного таргета: {sum(y_pred_default_class)/len(y_pred_default_class)}")
```

Оптимальный порог: 0.352633456530546



ROC-AUC: 0.8151223715960014, Gini: 0.6302447431920029  
 Распределение исходного таргета: 0.03332222592469177  
 Распределение предсказанного таргета: 0.32255914695101634

### Промежуточный итог:

- Когда известно описание и природа признаков, тогда к генерации признаков можно подойти осознано, если о признаках ничего не известно, как в нашем случае, тогда можно рассмотреть следующую генерацию: расчет отношения и произведения двух признаков, возведение в степень и т.д.
- Выбраны топ N признаков которые имеют наибольшее влияние на модель - feature importance и топ N признаков имеющих наибольшую корреляцию с таргетом. Произведена генерация отношений и произведений между выбранными признаками.
- Оценка модели, обученной на сгенерированных признаках показала хуже скор, чем без сгенерированных признаков.
- Генерация признаков в данном подходе не показала прирост к скору. В качестве развития возможно рассмотреть генерации по каждому признаку и прогон новых признаков через модель частями, чтобы не сильно увеличивать признаковое пространство

### Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Gini на test	Доля предсказанных дефолтов
1	Базовая модель	Простая модель	0.8794	0.8464	0.6929	0.0056
2	Модель №1 + балансировка классов	Учитываем в модели дисбаланс классов	0.8742	0.8488	0.6976	0.1059

3	Модель №2 + оптимальный threshold	Определяем оптимальный порог (threshold) для принятия решения о банкротстве. Модель не изменялась	0.8742	0.8488	0.6976	0.2235
4	Модель №3 + выделение категориальных признаков	Признаки с малым кол-ом уникальных значений переведены в тип категориальные	0.8814	0.8518	0.7037	0.2509
5	Модель №4 + генерация признаков	Генерация произведений и отношений между выбранным Топ признаков	0.8646	0.8151	0.6302	0.3225

## Исключение признаков не влияющих на таргет

Идея такая:

- Строим корреляцию для всех признаков
- Убираем признаки которые сильно корелируют с другими. Убираем признаки которые очень слабо корелируют с целевым таргетом

Ввод [89]:

```
%time
# Формирование матрицы корреляции
matrix_corr = train_nbki_df[feature_columns + [target_column]].corr()
matrix_corr.shape
```

Wall time: 1.22 s

Out[89]:

(128, 128)

Ввод [103]:

```
# Оставляем признаки у которых есть корреляция с целевой переменой (таргетом)
limit_corr_level = 0.01
target_corr = abs(matrix_corr[target_column])
features_target_corr = list(set(target_corr[target_corr > limit_corr_level].index) - set([target_column]))
print(f"Выбрано {len(features_target_corr)} признаков")
```

Выбрано 103 признаков

Ввод [104]:

```
# Убираем признаки с высоким уровнем корреляции между другими признаками
limit_corr_level = 0.95
select_features = []
skip_features = set([])

for feat in features_target_corr:
    if feat in skip_features:
        continue
    select_features.append(feat)
    skip_features |= set(matrix_corr[abs(matrix_corr[feat]) > limit_corr_level][feat].index)
print(f"Выбрано {len(select_features)} признаков")
```

Выбрано 94 признаков

Ввод [105]:

```
# Определяем список категориальных признаков из выбранных
select_categorical_columns = list(set(categorical_columns) & set(select_features))
print(f"Кол-во категориальных признаков ({len(select_categorical_columns)}) из выбранных ({len(select_
```

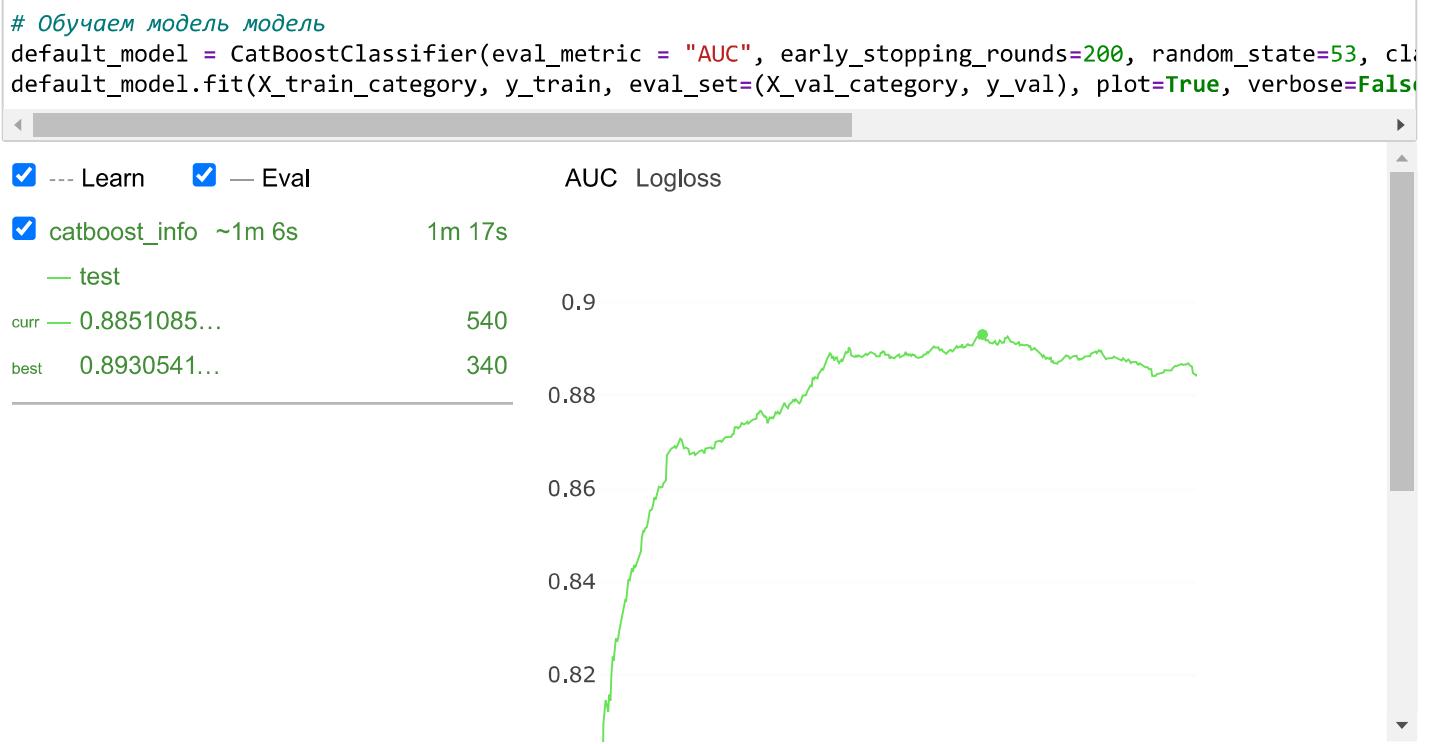
Кол-во категориальных признаков (39) из выбранных (94) признаков

Ввод [106]:

```
X_train_category = X_train[select_features].copy()
X_val_category = X_val[select_features].copy()
X_test_category = X_test[select_features].copy()

X_train_category[select_categorical_columns] = X_train_category[select_categorical_columns].astype(str)
X_val_category[select_categorical_columns] = X_val_category[select_categorical_columns].astype(str)
X_test_category[select_categorical_columns] = X_test_category[select_categorical_columns].astype(str)
```

Ввод [94]:



## Ввод [95]:

```
# Делаем предсказание на тестовых данных
y_pred_default = default_model.predict_proba(X_test_category)[:,1]

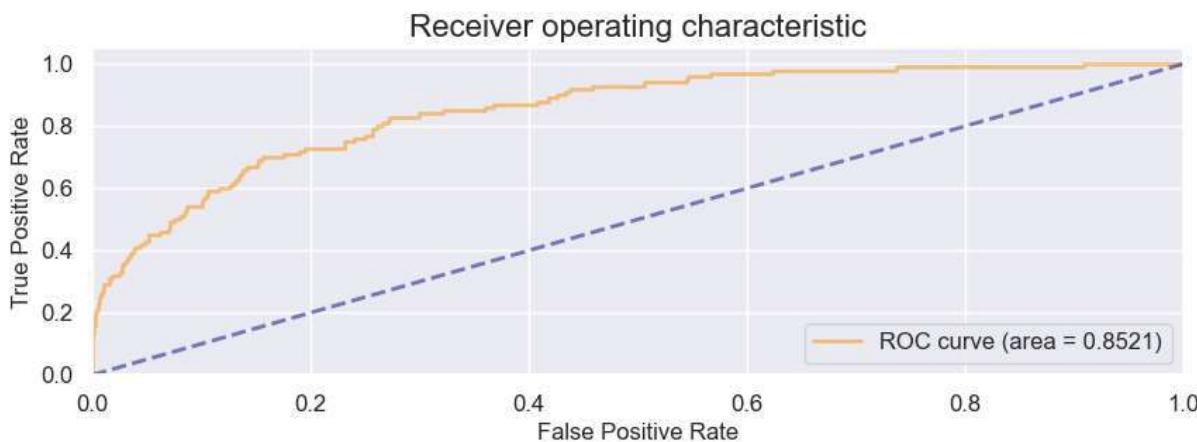
# Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=y_pred_default)
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")

# Формируем итоговое предсказание с учетом выбранного порога
y_pred_default_class = (y_pred_default > best_thresh).astype(int)

# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test, y_pred=y_pred_default)
print(f"ROC-AUC: {roc_auc}, Gini: {calc_gini(roc_auc)}")

print(f"Распределение исходного таргета: {sum(y_test)/len(y_test)}")
print(f"Распределение предсказанного таргета: {sum(y_pred_default_class)/len(y_pred_default_class)}")
```

Оптимальный порог: 0.23146345632993565



ROC-AUC: 0.8520751465012065, Gini: 0.704150293002413  
 Распределение исходного таргета: 0.03332222592469177  
 Распределение предсказанного таргета: 0.29090303232255915

## Промежуточный итог:

- Убраны признаки которые сильно корелируют с другими признаками ( $>0.95$ ), и убраны признаки которые очень слабо корелируют с целевым таргетом ( $\leq 0.01$ )
- Чистка признаков показала небольшой прирост к итоговому скору на тестовых данных. Также модель дальше не переобучается и на валидации показала скор выше

## Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Gini на test	Доля предсказанных дефолтов
1	Базовая модель	Простая модель	0.8794	0.8464	0.6929	0.0056
2	Модель №1 + балансировка классов	Учитываем в модели дисбаланс классов	0.8742	0.8488	0.6976	0.1059
3	Модель №2 + оптимальный threshold	Определяем оптимальный порог (threshold) для принятия решения о банкротстве. Модель не изменилась	0.8742	0.8488	0.6976	0.2235
4	Модель №3 + выделение категориальных признаков	Признаки с малым кол-ом уникальных значений переведены в тип категориальные	0.8814	0.8518	0.7037	0.2509
5	Модель №4 + генерация признаков	Генерация произведений и отношений между выбранным Топ признаков	0.8646	0.8151	0.6302	0.3225

6	Модель №4 + чистка признаков	Исключены признаки слабо влияющие на таргет и признаки которые сильно коррелируют между собой	0.8930	0.8520	0.7041	0.2909
---	---------------------------------	--	--------	--------	--------	--------

Лучшее качество показала Модель №6, которая включает: балансировку классов + выбор threshold + выделение категориальных признаков + генерацию и чистку признаков

## Подбор гиперпараметров

## Ввод [96]:

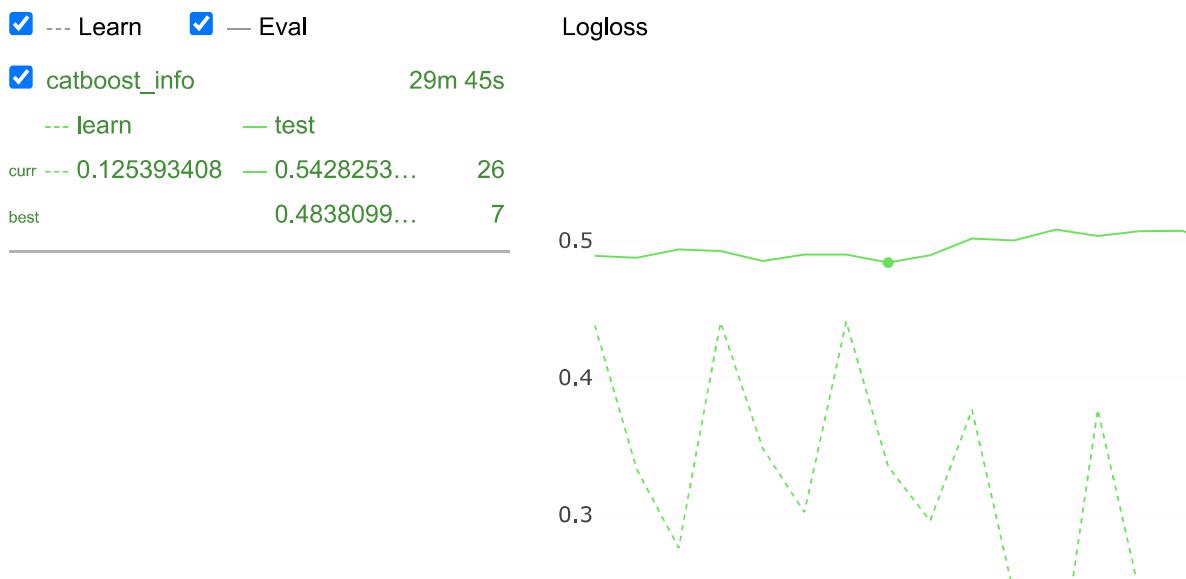
```
# Смотрим текущие значения параметров модели  
default model.get_all_params()
```

Out[96]:

```
{'nan_mode': 'Min',
 'eval_metric': 'AUC',
 'combinations_ctr': ['Borders:CtrBorderCount=15:CtrBorderType=Uniform:TargetBorderCount=1:TargetBorderType=MinEntropy:Prior=0/1:Prior=0.5/1:Prior=1/1',
 'Counter:CtrBorderCount=15:CtrBorderType=Uniform:Prior=0/1'],
 'iterations': 1000,
 'sampling_frequency': 'PerTree',
 'fold_permutation_block': 0,
 'leaf_estimation_method': 'Newton',
 'od_pval': 0,
 'counter_calc_method': 'SkipTest',
 'grow_policy': 'SymmetricTree',
 'penalties_coefficient': 1,
 'boosting_type': 'Plain',
 'model_shrink_mode': 'Constant',
 'feature_border_type': 'GreedyLogSum',
 'ctr_leaf_count_limit': 18446744073709551615,
 'bayesian_matrix_reg': 0.10000000149011612}.
```

Ввод [101]:

```
%time
parameters = {'l2_leaf_reg': [2, 3, 4],
              'depth': [4, 6, 8],
              'learning_rate' : [0.01, 0.04, 0.07],
              }
grid_search_model = CatBoostClassifier(early_stopping_rounds=200, random_state=53, class_weights=class_
grid search model.grid search(param grid=parameters, X=X train category, y=y train, stratified=True, p_
```



**Промежуточный итог:**

- Выбранные признаки Catboost по умолчанию оптимально подходят к решению задачи

**Формируем прогноз для тестовых данных (NBKI\_y\_test.csv)**

Изначально проводим новое обучение модели, но уже выделенные тестовые данные (из всей выборки) включаем в обучающую выборку

А в качестве тестовых данных будут использоваться данные из NBKI\_y\_test.csv

Ввод [ ]:

```
# X_train_category = X_train[select_features].copy()
# X_val_category = X_val[select_features].copy()
# X_test_category = X_test[select_features].copy()

# X_train_category[select_categorical_columns] = X_train_category[select_categorical_columns].astype(str)
# X_val_category[select_categorical_columns] = X_val_category[select_categorical_columns].astype(str)
# X_test_category[select_categorical_columns] = X_test_category[select_categorical_columns].astype(str)
```

Ввод [116]:

```
# Заново обучаем модель, которая по итогам экспериментов получилась - это Модель №4 (балансировка классов)
print(f"Кол-во признаков попадающих под тип категориальный: {len(categorical_columns)}")

train_df, val_df = train_test_split(train_nbki_df[feature_columns + [target_column]], test_size=0.05, random_state=42)
test_true_df = test_nbki_df[feature_columns + [target_column]].copy()

X_train = train_df[select_features]
y_train = train_df[target_column]
X_val = val_df[select_features]
y_val = val_df[target_column]
X_test_true = test_true_df[select_features]
y_test_true = test_true_df[target_column]

X_train[select_categorical_columns] = X_train[select_categorical_columns].astype(int)
X_val[select_categorical_columns] = X_val[select_categorical_columns].astype(int)
X_test_true[select_categorical_columns] = X_test_true[select_categorical_columns].astype(int)

X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test_true.shape, y_test_true.shape
```

Кол-во признаков попадающих под тип категориальный: 54

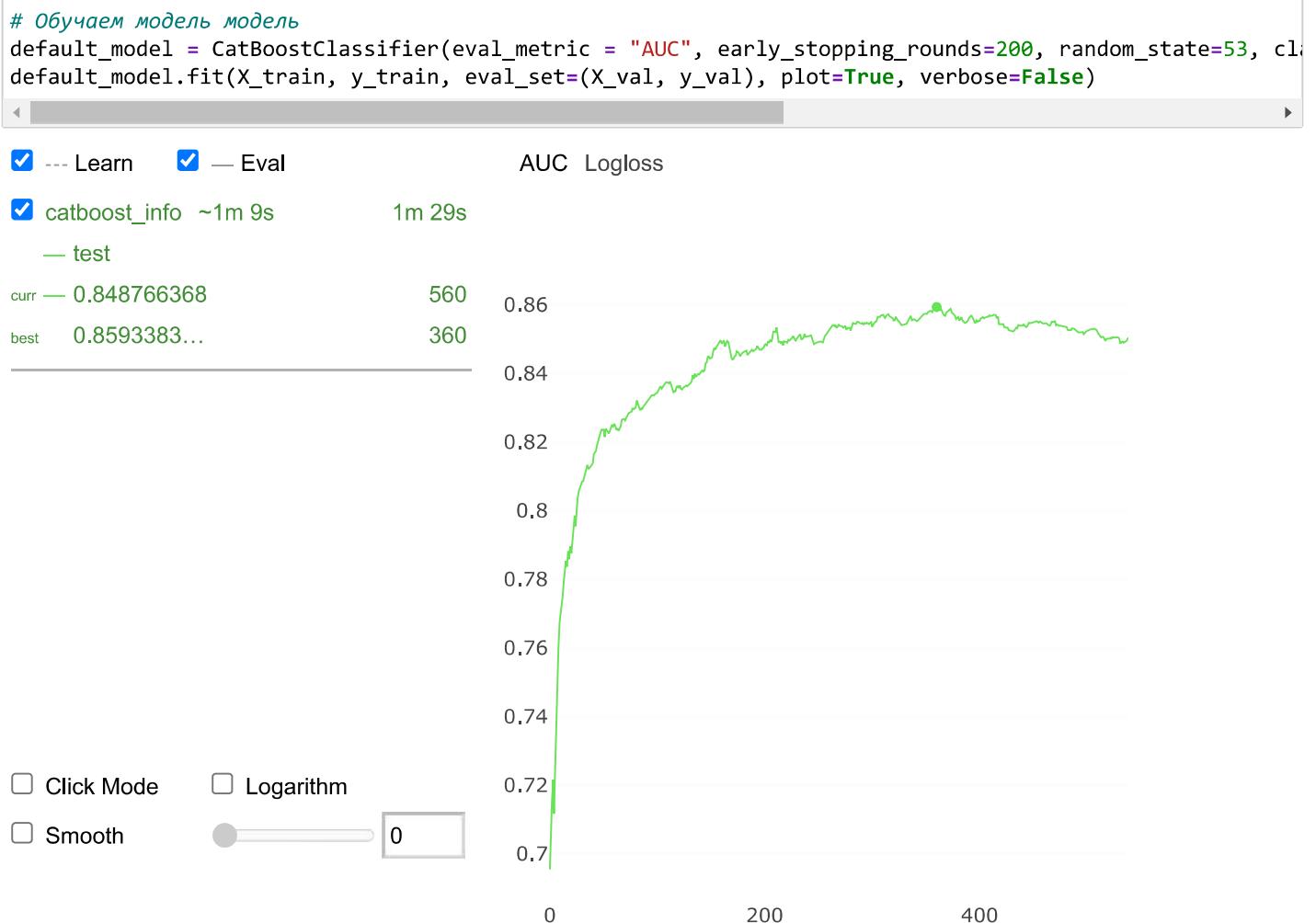
```
D:\_Work\_Projects\_Conda\DL52\lib\site-packages\pandas\core\frame.py:3641: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-a-copy)
    self[k1] = value[k2]
```

Out[116]:

```
((28506, 94), (28506,), (1501, 94), (1501,), (29993, 94), (29993,))
```

Ввод [118]:



Out[118]:

&lt;catboost.core.CatBoostClassifier at 0x21109bd91c8&gt;

Ввод [122]:

```
# Делаем предсказание на тестовых данных
y_pred_default = default_model.predict_proba(X_test_true)[:,1]

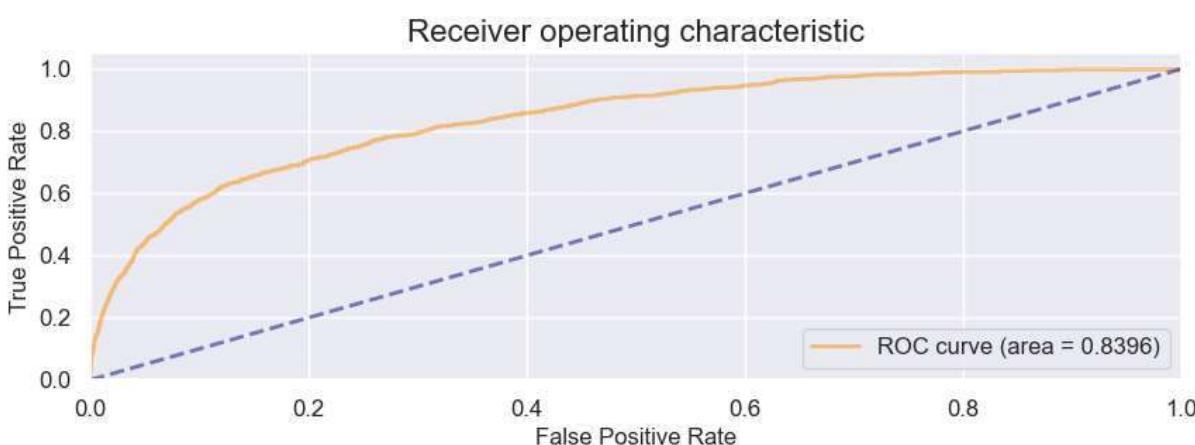
# Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test_true, y_score=y_pred_default)
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")

# Формируем итоговое предсказание с учетом выбранного порога
y_pred_default_class = (y_pred_default > best_thresh).astype(int)

# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_true, y_pred=y_pred_default)
print(f"ROC-AUC: {roc_auc}, Gini: {calc_gini(roc_auc)}")

print(f"Распределение исходного таргета: {sum(y_test_true)/len(y_test_true)}")
print(f"Распределение предсказанного таргета: {sum(y_pred_default_class)/len(y_pred_default_class)}")
```

Оптимальный порог: 0.2505765177415021



ROC-AUC: 0.8396213856596507, Gini: 0.6792427713193014  
 Распределение исходного таргета: 0.03127396392491581  
 Распределение предсказанного таргета: 0.2760644150301737

Ввод [124]:

```
# Формируем ответный файл прогноза для тестовой выборки - "NBKI_y_test_pred.csv"
test_nbki_df["default"] = y_pred_default_class
test_nbki_df.reset_index()[["Unnamed: 0", "default"]].rename(columns={"Unnamed: 0": ""}).to_csv(PATH_DA
```

## Отчет

Выполнено:

### 1. Анализ данных

- Проведен анализ данных
- Обработаны пропуски и неверные значения
- Анализ значимости признаков. Построение матрицы корреляции
- Анализ значений близких к константе

### 2. Разработка моделей

- Построение базовой модели
- Проведена балансировка классов
- Выбор порога (threshold) для достижения заданных метрик
- Генерация новых признаков (не улучшило скор)

