

Тестовое задание - анализ данных

Организация выдает займы физическим лицам. Информация о ранее выданных организацией займах в файле [new_training_data_31_08_2022.csv](https://dl.dropboxusercontent.com/s/6tg4aa9kt1y3dar/new_training_data_31_08_2022.csv) (https://dl.dropboxusercontent.com/s/6tg4aa9kt1y3dar/new_training_data_31_08_2022.csv). Файл [all_reject_data.csv](https://dl.dropboxusercontent.com/s/tvno43de29nu1rb/all_reject_data.csv) (https://dl.dropboxusercontent.com/s/tvno43de29nu1rb/all_reject_data.csv) содержит данные о клиентах, оставлявших заявки на заем, но в итоге не взявших его по причине отказа финансовой организации или нежеланию самого клиента.

Необходимо

- 1 - выполнить анализ и охарактеризовать клиентский портфель организации
- 2 - построить базовую модель прогнозирования банкротства, одобряющую не менее 35% клиентов при банкротстве среди одобренных не выше 15%.
- 3 - подготовить рекомендации и предложения по изменению признакового пространства, использованию внешних данных и иному развитию базовой модели.

Описание признакового пространства

Признак	Описание
age	Возраст заемщика
lastcredit	Время в днях, которое прошло с момента открытия последнего кредитного продукта
time_to_lastcredit_closeddt	Время в днях, которое прошло с момента закрытия последнего микрокредита (если есть активные кредиты, эта переменная будет равна 0)
close_loan_median	Медиана, взятая по количеству дней между закрытием предыдущего и следующего микрокредита (считается по всем последовательно открытым микрозаймам), т.е. сколько в среднем проходит времени между закрытием предыдущего и следующего микрокредита
open_loan_median	Медиана, взятая по количеству дней между открытием предыдущего и следующего микрокредита (считается по всем последовательно открытым микрозаймам), т.е. сколько в среднем проходит времени между открытием предыдущего и следующего микрокредита
is_active_100	Количество активных кредитов, открытых за все время
isnt_active_100	Доля не возвращенных кредитов относительно всех кредитов, взятых за все время
is_lost_100	Невозвращенные кредиты, открытые за последний месяц (например, переданные по цессии)
micro_loans_active_100	Активная сумма микрокредитов, открытых за всё время
is_active_12	Количество всех активных кредитов, открытых за последние 12 месяцев
open_sum_12	Активная сумма кредитов, взятых за последние 12 месяцев
isnt_active_12	Количество закрытых кредитов, которые были открыты за последние 12 месяцев
is_lost_12	Невозвращенные кредиты, открытые за последние 12 месяцев
overdue_loans_12	Количество просроченных кредитов, открытых за последние 12 месяцев
micro_loans_active_12	Активная сумма микрокредитов, открытых за последние 12 месяцев
is_active_3	Количество всех активных кредитов, открытых за последние 3 месяца
open_sum_3	Активная сумма кредитов, взятых за последние 3 месяца
isnt_active_3	Количество закрытых кредитов, которые были открыты за последние 3 месяца
is_lost_3	Невозвращенные кредиты, открытые за последние 3 месяца
overdue_loans_3	Количество просроченных кредитов, открытых за последние 3 месяца
micro_loans_active_3	Активная сумма микрокредитов, открытых за последние 3 месяца
is_active_1	Количество всех активных кредитов, открытых за последний месяц
open_sum_1	Активная сумма кредитов, взятых за последний месяц
isnt_active_1	Количество закрытых кредитов, которые были открыты за последний месяц
is_lost_1	Невозвращенные кредиты, открытые за последний месяц (например, переданные по цессии)
micro_loans_active_1	Активная сумма микрокредитов, открытых за последний месяц
ratio_all_microloans_3_to_12	Отношение количества микрокредитов, взятых за последние 3 месяца, к количеству микрокредитов, взятых за последние 12 месяцев
ratio_overdue_loans_3_to_12	Отношение количества просроченных микрокредитов, взятых за последние 3 месяца, к количеству просроченных микрокредитов, взятых за последние 12 месяцев
ratio_history_100	Доля не возвращенных кредитов относительно всех кредитов, взятых за все время
ratio_history_12	Доля не возвращенных кредитов относительно всех кредитов, взятых за последние 12 месяцев
fraction_last_x_12	Доля кредитов, взятых за последние 12 месяцев, относительно всех кредитов истории
ratio_history_3	Доля не возвращенных кредитов относительно всех кредитов, взятых за последние 3 месяца
fraction_last_x_3	Доля кредитов, взятых за последние 3 месяца, относительно всех кредитов истории
ratio_history_1	Доля не возвращенных кредитов относительно всех кредитов, взятых за последний месяц
fraction_last_x_1	Доля кредитов, взятых за последний месяц, относительно всех кредитов истории
mean_delay_100_with_lag	Средняя просрочка за всё время (с лагом по времени в 2 месяца)
mean_delay_12_with_lag	Средняя просрочка за последние 12 месяцев (с лагом по времени в 2 месяца)
mean_delay_3_with_lag	Средняя просрочка за последние 3 месяца (с лагом по времени в 2 месяца)
mean_delay_1_with_lag	Средняя просрочка за последний месяц (с лагом по времени в 2 месяца)
ratio_mean_delay_3_to_12	Отношение средней просрочки за последние 3 месяца к средней просрочке за последние 12 месяцев (в днях, с лагом по времени в 2 месяца)
count_all_credits	Количество всех кредитов в истории
ratio_pattern_len_to_pattern_1	Отношение количества платежей в платежном паттерне к общему количеству запланированных платежей на данный момент
ratio_pattern_len_to_pattern_2	Отношение количества просрочек в 0-5 дней в платежном паттерне к общему количеству запланированных платежей на данный момент
ratio_pattern_len_to_pattern_3	
ratio_pattern_len_to_pattern_4	
ratio_pattern_len_to_pattern_bad_len	Отношение количества символов сильной просрочки (> 60 дней) в платежном паттерне к общему количеству символов в строке
last_microloan_openeddt	Время в днях, которое прошло с момента открытия последнего микрокредита
is_type_credit_card_100	Количество кредитов типа "кредитная карта", открытых за всё время
is_type_consumer_100	Количество кредитов типа "потребительский кредит", открытых за всё время
is_type_micro_100	Количество кредитов типа "микрокредит", открытых за всё время
is_active_type_credit_card_100	Количество активных кредитов, открытых за всё время с типом займа "кредитная карта"
is_active_type_consumer_100	Количество активных кредитов, открытых за всё время с типом займа "потребительский кредит"
is_active_type_micro_100	Количество активных кредитов, открытых за всё время с типом займа "микрокредит"
is_type_credit_card_12	Количество кредитов типа "кредитная карта", открытых за последние 12 месяцев
is_type_consumer_12	Количество кредитов типа "потребительский кредит", открытых за последние 12 месяцев
is_type_micro_12	Количество кредитов типа "микрокредит", открытых за последние 12 месяцев
is_active_type_credit_card_12	Количество активных кредитов, открытых за последние 12 месяцев с типом займа "кредитная карта"
is_active_type_consumer_12	Количество активных кредитов, открытых за последние 12 месяцев с типом займа "потребительский кредит"
is_active_type_micro_12	Количество активных кредитов, открытых за последние 12 месяцев с типом займа "микрокредит"
is_type_credit_card_3	Количество кредитов типа "кредитная карта", открытых за последние 3 месяца
is_type_consumer_3	Количество кредитов типа "потребительский кредит", открытых за последние 3 месяца

Признак	Описание
is_type_micro_3	Количество кредитов типа "микрокредит", открытых за последние 3 месяца
is_active_type_credit_card_3	Количество активных кредитов, открытых за последние 3 месяца с типом займа "кредитная карта"
is_active_type_consumer_3	Количество активных кредитов, открытых за последние 3 месяца с типом займа "потребительский кредит"
is_active_type_micro_3	Количество активных кредитов, открытых за последние 3 месяца с типом займа "микрокредит"
is_type_credit_card_1	Количество кредитов типа "кредитная карта", открытых за последние 12 месяцев
is_type_consumer_1	Количество кредитов типа "потребительский кредит", открытых за последний месяц
is_type_micro_1	Количество кредитов типа "микрокредит", открытых за последний месяц
is_active_type_credit_card_1	Количество активных кредитов, открытых за последние 12 месяцев с типом займа "кредитная карта"
is_active_type_consumer_1	Количество активных кредитов, открытых за последний 1 месяц с типом займа "потребительский кредит"
is_active_type_micro_1	Количество активных кредитов, открытых за последний 1 месяц с типом займа "микрокредит"
overall_worst_overdue_state_12	

Требование к модели

Построить базовую модель прогнозирования банкротства:

- одобряющую не менее 35% клиентов при банкротстве среди одобренных не выше 15%.

Принимаем допущение, что два списка кому дали заем и кому не дали это разные люди.

Если "таргет" в файле кому дали заем - это факт банкротства, тогда задача звучит следующим образом:

- Общее число людей: $106k+42k = 148k$
- Есть список кому отказали (или сами отказались): $106k$ (72%)
- Есть список кому одобрили: $42k$ (28%)
- Из тех кому одобрили ($42k$), часть банкроты: $11k$ (26%)

Целевая задача:

- сможет повысить % кому одобрили: с 28% прийти к 35%
- сможет понизить % банкротов среди тех кому одобрили: с 26% прийти к 15% Т.е. надо ослабить ограничение для выдачи займов, чтобы большему числу людей можно было выдать кредит, но при этом процент банкротов уменьшился.

В рамках задания требуется построить модель предсказывания банкротов. Для этого будут использованы данные о банкротах, которые в самом начале будут разбиты на три части train/test/val, при чем тестовые данные будут использоваться только в качестве подсчета итоговых метрик.

Загрузка данных

```
Ввод [44]: import pandas as pd
import numpy as np

import os

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

from sklearn.manifold import TSNE
from sklearn.preprocessing import StandardScaler

from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from imblearn.under_sampling import TomekLinks

import shap
from ydata_profiling import ProfileReport

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

```
Ввод [45]: PATH = "datasets/"
train_filename = PATH + "new_training_data_31_08_2022.csv"
all_reject_filename = PATH + "all_reject_data.csv"
```

```
Ввод [46]: # Список кому не выдали заем
reject_credit_df = pd.read_csv(all_reject_filename, index_col=False)
reject_credit_df["target"] = 0
reject_credit_df["accept"] = 0
# Список кому выдали заем
accept_credit_df = pd.read_csv(train_filename, index_col=False)
accept_credit_df["accept"] = 1

reject_credit_df.shape, accept_credit_df.shape
```

```
Out[46]: ((106511, 76), (42529, 76))
```

```
Ввод [47]: train_df = pd.concat([reject_credit_df, accept_credit_df])
train_df.shape
```

```
Out[47]: (149040, 76)
```

Первичный анализ

Ввод [48]: `train_df.describe()`

Out[48]:

	Unnamed: 0	age	lastcredit	time_to_lastcredit_closesdt	close_loan_median	open_loan_median	is_active_100	isnt_active_100	is_lost_100	micro_loans_active_100	...	is_l
count	149040.000000	149040.000000	149040.000000	149040.000000	148893.000000	148893.000000	149040.000000	149040.000000	149040.000000	149040.000000	1.490400e+05	...
mean	34199.910185	33.363580	29.137118	307.682783	3.09175	1733.474717	13.387245	40.363808	1.986520	1.578954e+05
std	26109.002324	8.903511	100.436634	3625.279321	49.63859	7659.052945	12.353431	69.986857	3.840825	2.205046e+05
min	0.000000	18.000000	1.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	...
25%	12821.000000	27.000000	3.000000	0.000000	0.00000	1.000000	4.000000	7.000000	0.000000	2.520000e+04
50%	26406.000000	32.000000	6.000000	0.000000	0.00000	3.000000	10.000000	20.000000	0.000000	8.304450e+04
75%	54387.250000	38.000000	17.000000	0.000000	0.00000	10.000000	19.000000	48.000000	2.000000	2.121732e+05
max	91647.000000	76.000000	5655.000000	44785.000000	2779.00000	44775.000000	153.000000	6743.000000	75.000000	2.199029e+07

8 rows × 76 columns

Параметр "ratio_sum_outstanding_to_open_sum" имеет значения +inf/-inf

Убираем бесконечности/неопределенности.

По описанию этот параметр представляет собой: "Отношение суммы просрочки по всем кредитам к сумме взятых кредитов за всю историю" Соответственно неопределенность может возникать, когда значения суммы просрочки сверх низкие

Ввод [49]: `min_values = min(train_df.describe().loc["min"])
max_values = max(train_df.describe().loc["max"])`

Out[49]: `(-inf, inf)`

Ввод [50]: `train_df["ratio_sum_outstanding_to_open_sum"].replace([np.inf, -np.inf], 0, inplace=True)`

Ввод [51]: `# Проверяем что -inf/inf нет в данных
min_values = min(train_df.describe().loc["min"])
max_values = max(train_df.describe().loc["max"])
assert not (np.isinf(max_values) and np.isinf(min_values)), "В данных есть inf"
min_values, max_values`

Out[51]: `(-18741.0, 31234690.0)`

Борьба с пропусками в данных

Ввод [52]: `# Смотрим в каких столбцах у нас пропуски данных
nan_contains = train_df.isna().sum()
nan_contains[nan_contains > 0]`

Out[52]:

close_loan_median	147
open_loan_median	147
ratio_all_microloans_3_to_12	621
ratio_overdue_loans_3_to_12	21707
ratio_history_100	27
ratio_history_12	514
fraction_last_x_12	27
ratio_history_3	2602
fraction_last_x_3	27
ratio_history_1	4267
fraction_last_x_1	27
ratio_mean_delay_3_to_12	5464
ratio_pattern_len_to_pattern_1	147
ratio_pattern_len_to_pattern_2	147
ratio_pattern_len_to_pattern_3	147
ratio_pattern_len_to_pattern_4	147
ratio_pattern_len_to_pattern_bad_len	147
ratio_sum_outstanding_to_open_sum	4180
	dtype: int64

Ввод [53]: `# Объем обучающей выборки 42тыс записей. Есть параметры, где пропуски составляют < 1% от всех записей.`

`# Такие записи можно безболезненно исключить из выборки. Но оставить те у которых пропусков более 1% записей`

Ввод [54]: `train_df = train_df.dropna(subset=['close_loan_median', 'open_loan_median', 'ratio_all_microloans_3_to_12', 'ratio_history_100', 'ratio_history_12', 'fraction_last_x_12'])`

Ввод [54]: `nan_contains = train_df.isna().sum()
nan_contains[nan_contains > 0]`

Out[54]:

ratio_overdue_loans_3_to_12	21080
ratio_history_3	2017
ratio_history_1	3661
ratio_mean_delay_3_to_12	4972
ratio_sum_outstanding_to_open_sum	4098
	dtype: int64

Остались параметры с NaN:

- ratio_overdue_loans_3_to_12 21080 (NaN)
- ratio_history_3 2017 (NaN)
- ratio_history_1 3661 (NaN)
- ratio_mean_delay_3_to_12 4972 (NaN)
- ratio_sum_outstanding_to_open_sum 4098 (NaN)

Для них попробуем определить оптимальные значения для замены пропусков

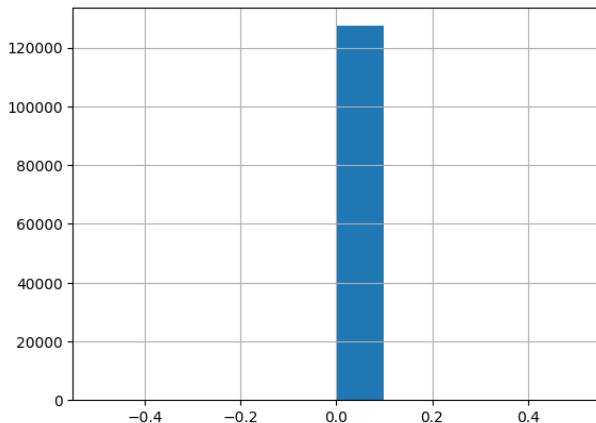
```
Ввод [55]: train_df[['ratio_overdue_loans_3_to_12', 'ratio_history_3', 'ratio_history_1', 'ratio_mean_delay_3_to_12', 'ratio_sum_outstanding_to_open_sum', ]].describe()
```

Out[55]:

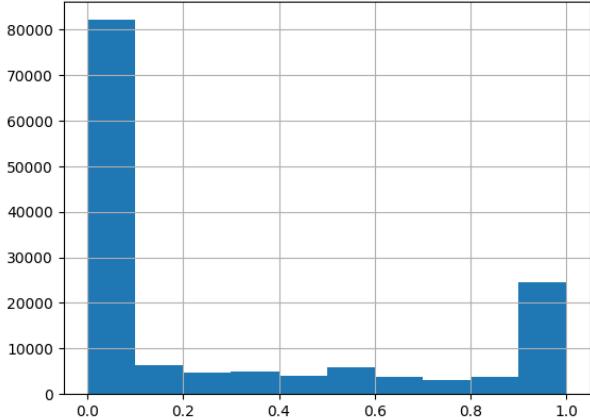
	ratio_overdue_loans_3_to_12	ratio_history_3	ratio_history_1	ratio_mean_delay_3_to_12	ratio_sum_outstanding_to_open_sum
count	127333.0	146396.000000	144752.000000	143441.000000	1.443150e+05
mean	0.0	0.064256	0.020111	0.290570	1.425404e+02
std	0.0	0.168297	0.099751	0.390858	7.538356e+03
min	0.0	0.000000	0.000000	0.000000	-1.874100e+04
25%	0.0	0.000000	0.000000	0.000000	0.000000e+00
50%	0.0	0.000000	0.000000	0.000000	1.666667e-01
75%	0.0	0.020000	0.000000	0.592994	1.041428e+00
max	0.0	1.000000	1.000000	1.000000	1.458958e+06

```
Ввод [56]: # Параметр ratio_overdue_loans_3_to_12 содержит константу 0 его надо исключить из рассмотрения при анализе константных параметров, пока заполняем константным значением нулем
train_df["ratio_overdue_loans_3_to_12"].hist()
print(train_df["ratio_overdue_loans_3_to_12"].unique())
train_df["ratio_overdue_loans_3_to_12"] = train_df["ratio_overdue_loans_3_to_12"].fillna(0)
```

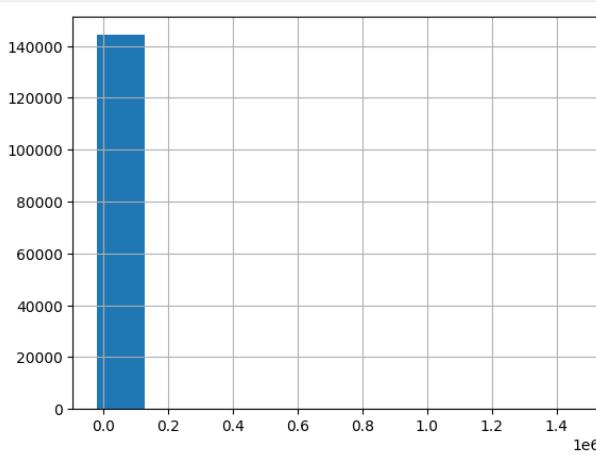
[nan 0.]



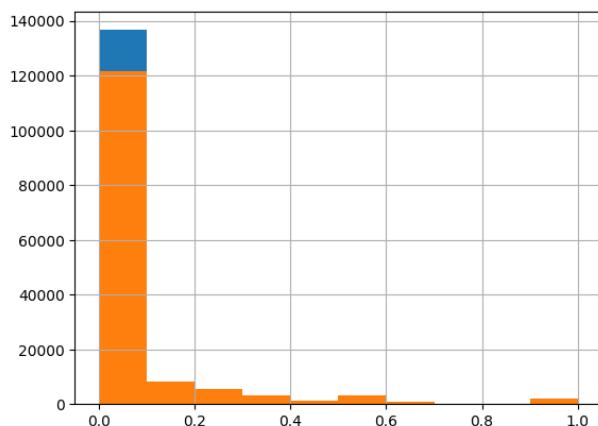
```
Ввод [57]: # Параметр ratio_mean_delay_3_to_12 представляет распределение от 0 до 1. Причем 0-ые значения гораздо больше, что означает что нельзя использовать среднее значение
train_df["ratio_mean_delay_3_to_12"].hist()
train_df["ratio_mean_delay_3_to_12"] = train_df["ratio_mean_delay_3_to_12"].fillna(train_df["ratio_mean_delay_3_to_12"].median())
```



```
Ввод [58]: # Параметр ratio_sum_outstanding_to_open_sum содержит практически все значения 0, этот признак близок к константному, и примененем на исключение, но пока заменим на медианное значение
train_df["ratio_sum_outstanding_to_open_sum"].hist()
train_df["ratio_sum_outstanding_to_open_sum"] = train_df["ratio_sum_outstanding_to_open_sum"].fillna(train_df["ratio_sum_outstanding_to_open_sum"].median())
```



```
Ввод [59]: # Параметр ratio_history_1 и ratio_history_3 это доли не возвращенных кредитов. Нулевых значений гораздо больше, что означает что нельзя использовать среднее значение
train_df["ratio_history_1"].hist()
train_df["ratio_history_3"].hist()
train_df["ratio_history_1"] = train_df["ratio_history_1"].fillna(train_df["ratio_history_1"].median())
train_df["ratio_history_3"] = train_df["ratio_history_3"].fillna(train_df["ratio_history_3"].median())
```

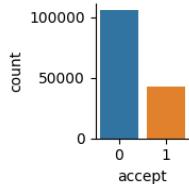


```
Ввод [60]: # Проверяем что в нашей выборке не осталось параметров с NaN
nan_contains = train_df.isna().sum()
assert len(nan_contains[nan_contains > 0]) == 0, "Остались параметры с NaN"
```

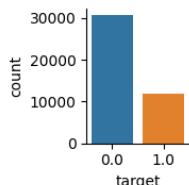
Проверка дисбаланса таргета

```
Ввод [61]: accept_count = train_df["accept"].value_counts()
print(f"Займ выдан {accept_count[1]} ({int(accept_count[1]/len(train_df)*100)}%) из {len(train_df)}")
sns.catplot(data=train_df, kind='count', x='accept', height=2)
plt.show()
# Баланс классов примерно 1 к 2, это уже как минимум не задача поиска аномалий, просто надо учитывать этот дисбаланс при обучении модели
target_count = train_df[train_df["accept"]==1]["target"].value_counts()
print(f"Банкротов {target_count[1]} ({int(target_count[1]/accept_count[1]*100)}%) среди тех кому выдали займ {accept_count[1]}")
# train_df[train_df["accept"]==1]["target"]
sns.catplot(data=train_df[train_df["accept"]==1], kind='count', x='target', height=2)
plt.show()
```

Займ выдан 42497 (28%) из 148413



Банкротов 11785 (27%) среди тех кому выдали займ 42497



Анализ значимости признаков

Проведем анализ значимости признаков:

- значения близкие к константе
- корреляция

Категориальных значений в данных нет, следовательно анализ через хи-квадрат выполнить не получится

```
Ввод [62]: # Исключаем из данных колонку с порядковым номером "Unnamed: 0" данный столбец неинформативен. Встречаются одинаковые номера для разных людей (разный возраст), поэтому удаляем
train_df = train_df.drop(columns="Unnamed: 0")
```

```
Ввод [63]: # определяем колонки таргетов и признаков
target_columns = ["target", "accept"]
feature_columns = train_df.columns.drop(target_columns)
```

```
Ввод [64]: # Ищем параметры с константными значениями
for column in feature_columns:
    count_unique_values = len(train_df[column].unique())
    if count_unique_values == 1:
        print(f"Constant column: {column}: {count_unique_values}")
        train_df = train_df.drop(columns=column)
train_df.shape
```

Constant column: overdue_loans_12: 1
Constant column: overdue_loans_3: 1
Constant column: ratio_overdue_loans_3_to_12: 1

Out[64]: (148413, 72)

```
Ввод [65]: # удаляем дубликаты в данных
train_df = train_df.drop_duplicates()
train_df.shape
```

```
Out[65]: (148020, 72)
```

Матрица корреляции

```
Ввод [66]: corr_matrix = train_df.corr()
corr_matrix_accept = train_df[train_df["accept"]==1].corr()
```

```
Ввод [67]: corr_matrix
```

```
Out[67]:
```

	age	lastcredit	time_to_lastcredit_closeddt	close_loan_median	open_loan_median	is_active_100	isnt_active_100	is_lost_100	micro_loans_active_100	is_act
age	1.000000	-0.022704	0.014347	0.009456	-0.034545	0.092945	0.056732	-0.012173	0.128261	0.0
lastcredit	-0.022704	1.000000	0.113537	0.033453	0.093424	-0.209915	-0.065950	0.160499	-0.160143	-0.2
time_to_lastcredit_closeddt	0.014347	0.113537	1.000000	0.030730	0.075809	-0.038707	-0.038866	-0.037090	-0.037429	-0.0
close_loan_median	0.009456	0.033453	0.030730	1.000000	0.116069	-0.059262	-0.032864	-0.014918	-0.043069	-0.0
open_loan_median	-0.034545	0.093424	0.075809	0.116069	1.000000	-0.209872	-0.125250	-0.095592	-0.153807	-0.1
...
is_active_type_micro_1	0.025651	-0.279710	-0.040135	-0.047868	-0.168833	0.793185	0.285974	-0.231205	0.627284	0.6
overall_worst_overdue_state_12	0.060084	0.221944	-0.020402	-0.028157	-0.166941	0.126805	0.233449	0.220716	0.141511	0.0
ratio_sum_outstanding_to_open_sum	0.008486	0.031406	-0.001081	0.002698	0.000316	-0.018190	-0.005441	0.014042	-0.013342	-0.0
target	0.034566	-0.058227	-0.006261	-0.013770	-0.051255	0.137640	0.041369	-0.093452	0.108371	0.1
accept	0.104003	-0.154662	-0.022140	-0.035098	-0.125352	0.364571	0.168649	-0.229314	0.330658	0.0

72 rows x 72 columns

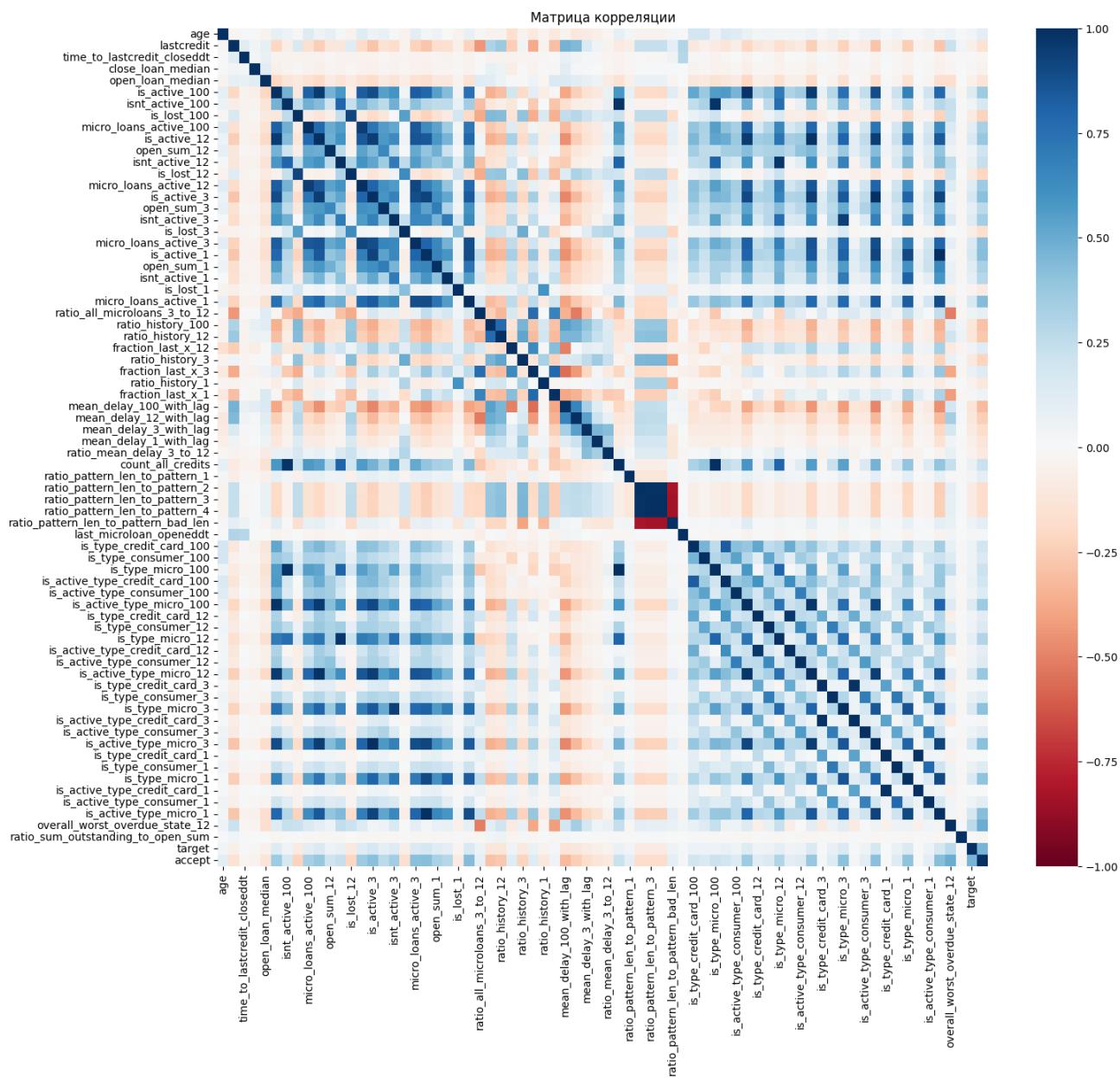
```

Ввод [68]: %%time
plt.figure(figsize=(16, 14))
heatmap = sns.heatmap(corr_matrix, vmin=-1, vmax=1, cmap='RdBu')
heatmap.set_title('Матрица корреляции');

Wall time: 654 ms

```

Out[68]: Text(0.5, 1.0, 'Матрица корреляции')

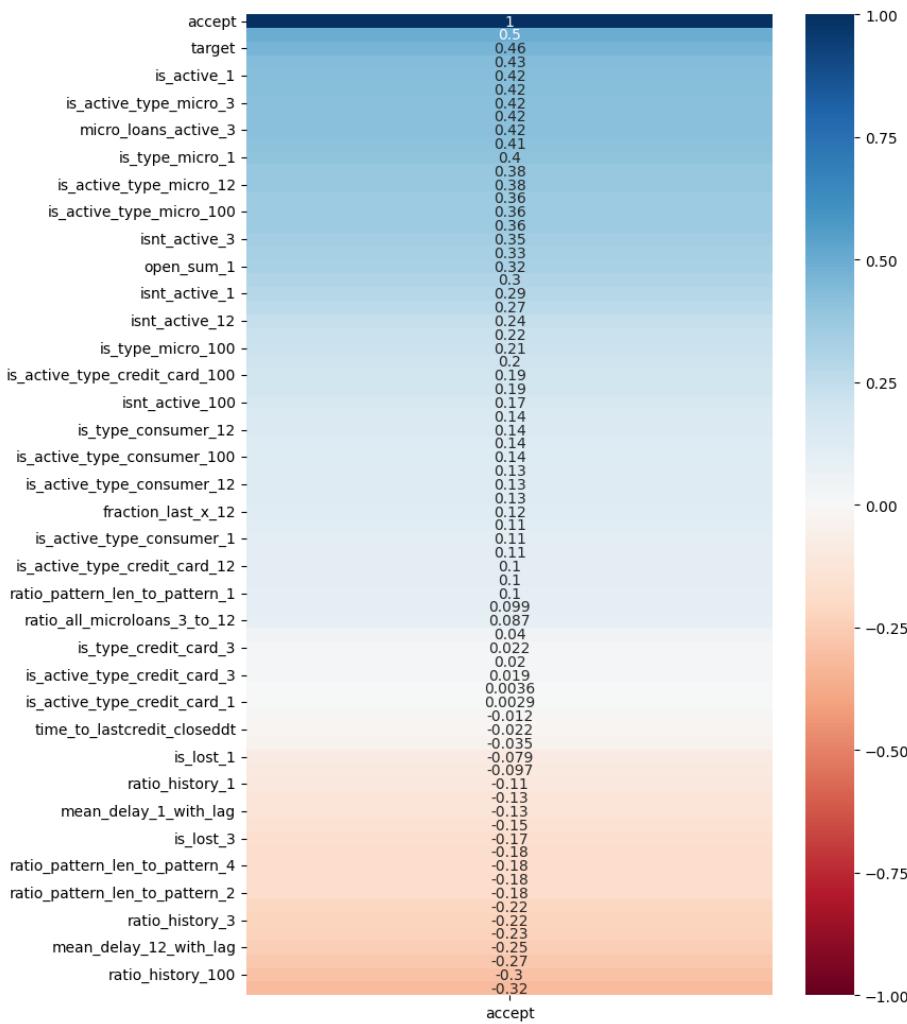


По матрице корреляции видно, что есть признаки, коррелирующие между собой. Коррелирующие признаки необходимо исключить из выборки.

Признаки коррелирующие с таргетом

```
Ввод [69]: # Признаки влияющие на выдачу займа
plt.figure(figsize=(8, 12))
heatmap = sns.heatmap(corr_matrix[['accept']].sort_values(by='accept', ascending=False), vmin=-1, vmax=1, annot=True, cmap='RdBu')
heatmap.set_title('Корреляция признаков с accept', fontdict={'fontsize':18}, pad=16);
```

Корреляция признаков с accept



```
Ввод [70]: # Топ признаков влияющие на выдачу займа
corr_matrix[corr_matrix['accept'].abs() > 0.3]['accept']
```

```
Out[70]: is_active_100          0.364571
micro_loans_active_100        0.330658
is_active_12                  0.379584
micro_loans_active_12        0.359512
is_active_3                   0.419726
isnt_active_3                0.348728
micro_loans_active_3         0.418686
is_active_1                   0.423291
open_sum_1                    0.320494
micro_loans_active_1          0.432750
mean_delay_100_with_lag      -0.315668
is_active_type_micro_100      0.364030
is_type_micro_12              0.303245
is_active_type_micro_12       0.379523
is_type_micro_3               0.412681
is_active_type_micro_3        0.420758
is_type_micro_1               0.404371
is_active_type_micro_1        0.422943
overall_worst_overdue_state_12 0.495858
target                         0.463443
accept                          1.000000
Name: accept, dtype: float64
```

Анализ признаков коррелирующих выдачей займов

Топ признаков влияющие на выдачу займа:

Признак	Корреляция	Описание
is_active_1	0.423296	Количество всех активных кредитов, открытых за последний 1мес.
is_active_100	0.364380	Количество всех активных кредитов, открытых за все время
is_active_12	0.379295	Количество всех активных кредитов, открытых за последние 12 мес.
is_active_3	0.419496	Количество всех активных кредитов, открытых за последний 3 мес.
isnt_active_3	0.348374	Количество всех активных кредитов, открытых за последние 3 месяца
is_type_micro_12	0.303220	Количество закрытых кредитов, которые были открыты за последние 3 месяца
is_type_micro_3	0.412443	Количество кредитов типа "микрокредит", открытых за последние 3 месяца
is_type_micro_1	0.404514	Количество кредитов типа "микрокредит", открытых за последние 1 месяца
micro_loans_active_100	0.330461	Активная сумма микрокредитов, открытых за всё время
micro_loans_active_12	0.359170	Активная сумма микрокредитов, открытых за последние 12 месяцев

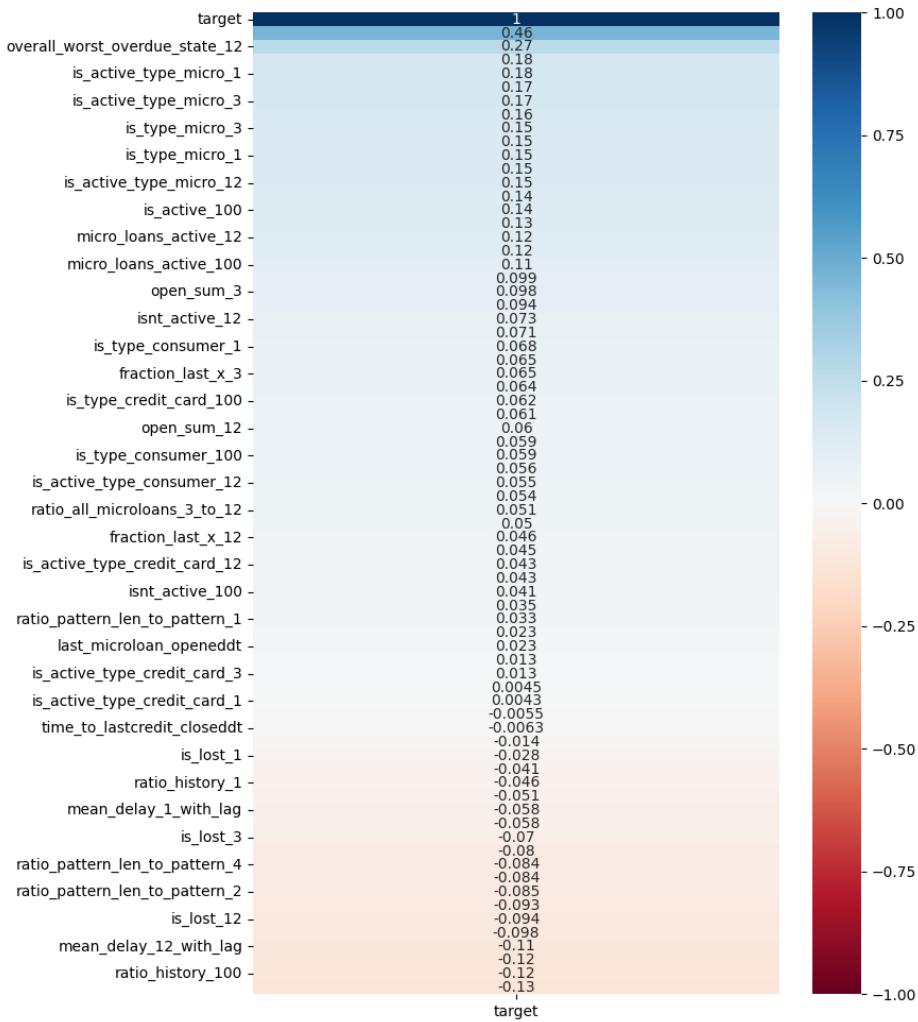
Признак	Корреляция	Описание
micro_loans_active_3	0.418282	Активная сумма микрокредитов, открытых за последние 3 месяца
micro_loans_active_1	0.432544	Активная сумма микрокредитов, открытых за последний 1 месяц
open_sum_1	0.320523	Активная сумма кредитов, взятых за последний месяц
is_active_type_micro_100	0.363796	Количество активных кредитов, открытых за все время с типом займа "микрокредит"
is_active_type_micro_12	0.379225	Количество активных кредитов, открытых за 12 месяцев с типом займа "микрокредит"
is_active_type_micro_3	0.420535	Количество активных кредитов, открытых за 3 месяцев с типом займа "микрокредит"
is_active_type_micro_1	0.422957	Количество активных кредитов, открытых за 1 месяц с типом займа "микрокредит"
overall_worst_overdue_state_12	0.492895	Для этого признака нет описания, но судя по названию, это общий наихудший статус по просроченным платежам за 12 месяцев. Чем лучше этот статус тем вероятнее выдадут займ. Чем хуже этот статус просрочки, тем вероятнее будет банкрот (положительная корреляция)
mean_delay_100_with_lag	-0.315969	Средняя просрочка за всё время (с лагом по времени в 2 месяца) Этот признак имеет отрицательную корреляцию, что логично чем меньше просрочки по прошлым кредитам, тем охотнее дадут новый займ

Итого:

- Очень сильно влияет кол-во открытых кредитов и кол-во закрытых кредитов, что логично если человек часто пользуется микрозаймами и всегда их закрывает, ему скорее всего одобрят и новый займ
- Если тип займа "микрокредит" тогда займ дадут с уверенностью

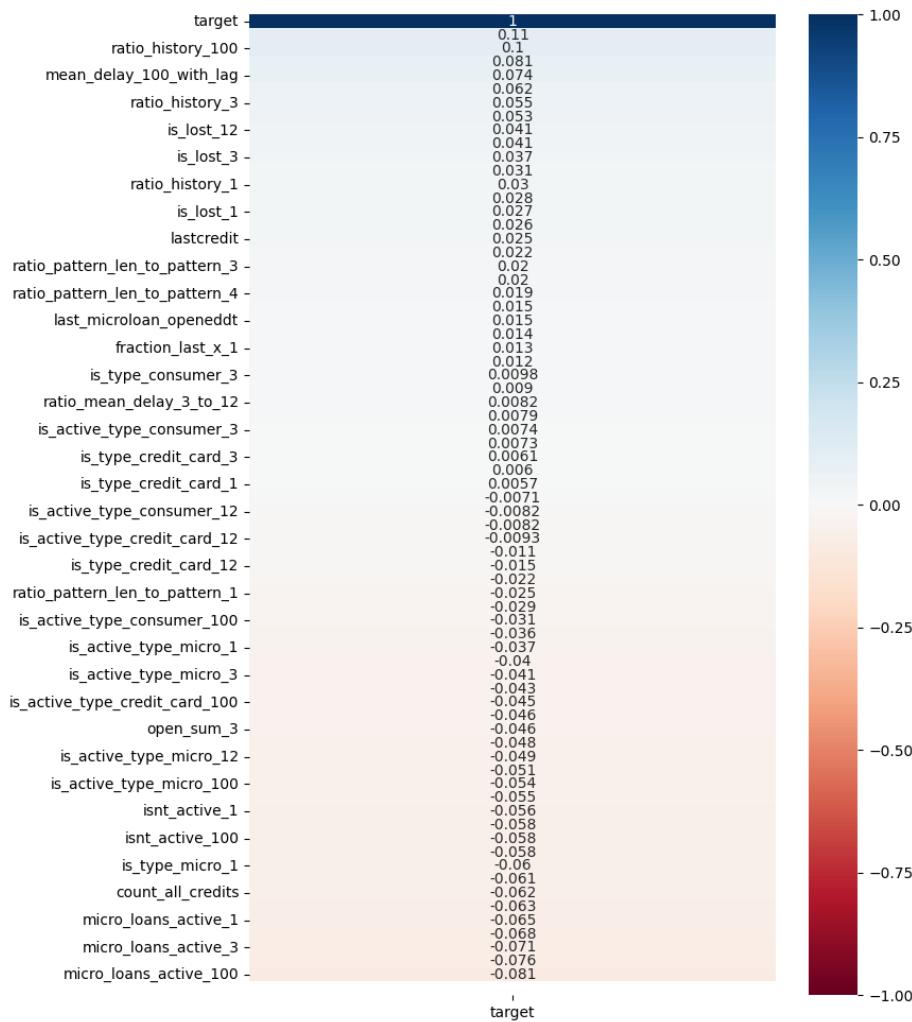
```
Ввод [71]: # Признаки коррелирующие с банкротством
plt.figure(figsize=(8, 12))
heatmap = sns.heatmap(corr_matrix[['target']].sort_values(by='target', ascending=False), vmin=-1, vmax=1, annot=True, cmap='RdBu')
heatmap.set_title('Корреляция признаков с target/банкротство', fontdict={'fontsize':18}, pad=16);
```

Корреляция признаков с target/банкротство



```
Ввод [72]: # Признаки коррелирующие с банкротством учитывая только выданные займы
plt.figure(figsize=(8, 12))
heatmap = sns.heatmap(corr_matrix_accept[['target']].sort_values(by='target', ascending=False), vmin=-1, vmax=1, annot=True, cmap='RdBu')
heatmap.set_title('Корреляция признаков с target/банкротство (только для выданных займов)', fontdict={'fontsize':18}, pad=16);
```

Корреляция признаков с target/банкротство (только для выданных займов)



```
Ввод [73]: # Топ признаков влияющие на факт банкротства:
corr_matrix_accept[corr_matrix_accept['target'].abs() > 0.07]['target']
```

```
Out[73]: micro_loans_active_100      -0.081207
micro_loans_active_12      -0.076477
micro_loans_active_3       -0.071073
ratio_history_100          0.104963
ratio_history_12            0.080794
mean_delay_100_with_lag    0.073886
overall_worst_overdue_state_12  0.108484
target                      1.000000
Name: target, dtype: float64
```

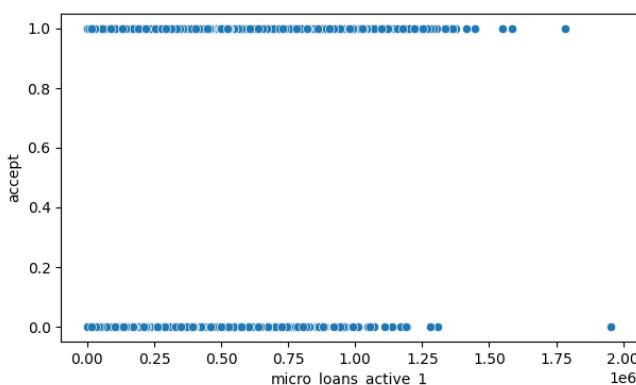
Анализ признаков коррелирующих с банкротством

Топ признаков влияющие на факт банкротства:

Признак	Корреляция	Type corr	Описание
overall_worst_overdue_state_12	0.105773	Pos corr	Для этого признака нет описания, но судя по названию, это общий наихудший статус по просроченным платежам за 12 месяцев. Чем хуже этот статус просрочки, тем вероятнее будет банкрот (положительная корреляция). Чем лучше этот статус тем вероятнее выдадут займ
ratio_history_100	0.104355	Pos corr	Доля не возвращенных кредитов относительно всех кредитов, взятых за все время. Чем больше доля не возвращенных всех кредитов, тем вероятнее будет банкрот (положительная корреляция)
ratio_history_12	0.080363	Pos corr	Чем больше доля не возвращенных кредитов за год, тем вероятнее будет банкрот (положительная корреляция)
mean_delay_100_with_lag	0.073888	Pos corr	Средняя просрочка за всё время (с лагом по времени в 2 месяца) Чем больше просрочка, тем вероятнее будет банкрот (положительная корреляция)
micro_loans_active_3	-0.072569	Neg corr	Активная сумма микрокредитов, открытых за последние 3 месяца
micro_loans_active_12	-0.077707	Neg corr	Активная сумма микрокредитов, открытых за последние 12 месяцев
micro_loans_active_100	-0.082489	Neg corr	Активная сумма микрокредитов, открытых за всё время. Чем меньше активная сумма микрокредитов, тем вероятнее будет банкрот (отрицательная корреляция).

Описание для параметров `micro_loans_active_*` не дает представление о сущности этого признака, что значит активная, это сколько человек сейчас должен по всем микрозаймам, или это показатель как человек активно гасит кредит. Судя по корреляции чем больше это значение тем меньше вероятность банкротства

```
Ввод [74]: # Выведем пример корреляции "micro_loans_active_1" с "accept"
plt.figure(figsize=(7,4))
sns.scatterplot(data=train_df, x='micro_loans_active_1', y='accept');
```



Ввод []:

Профилирование данных

Учитывая, что признаков в данных довольно много, профилирование будем делать частями

```
Ввод [75]: %time
# Профилирование данных. Много признаков, поэтому отчет сохраним во внешний html файл, чтобы не нагружать jupyter-ноутбук
profile_report_filename = "ProfileReport_train.html"
if not os.path.exists(profile_report_filename):
    # Время генерации 10-15 минут
    profile = ProfileReport(train_df)
    profile.to_file(profile_report_filename)
```

Wall time: 999 µs

Признаки с большим кол-вом нулей

В данных встречаются признаки с большим кол-вом нулей, например:

- time_to_lastcredit_closeddt

`time_to_lastcredit_closeddt`
Real number (ℝ)

ZEROS

Distinct	1091
Distinct (%)	0.7%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	220.99228

Minimum	0
Maximum	44785
Zeros	136861
Zeros (%)	92.5%
Negative	0
Negative (%)	0.0%
Memory size	2.3 MiB



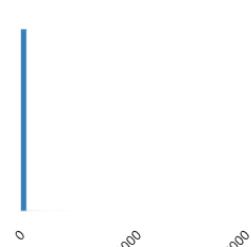
- close_loan_median

`close_loan_median`
Real number (ℝ)

SKEWED ZEROS

Distinct	805
Distinct (%)	0.5%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	3.1055465

Minimum	0
Maximum	2779
Zeros	143998
Zeros (%)	97.3%
Negative	0
Negative (%)	0.0%
Memory size	2.3 MiB



- is_lost_1

`is_lost_1`
Real number (ℝ)

ZEROS

Distinct	14
Distinct (%)	< 0.1%
Missing	0
Missing (%)	0.0%
Infinite	0
Infinite (%)	0.0%
Mean	0.11024862

Minimum	0
Maximum	24
Zeros	137087
Zeros (%)	92.6%
Negative	0
Negative (%)	0.0%
Memory size	2.3 MiB



- last_microloan_openeddt



Такие признаки, по-хорошему необходимо исключить из признакового пространства. Однако сначала построим бейзлайн модель и определим какие признаки действительно слабо влияют на таргет

Определяем тренировочную и тестовые выборки

- В качестве модели будем использовать CatBoost поэтому специально нормализацию данных делать не придется (для деревьев нормализация не нужна)
- Данные разделяются на три группы:

- займ не дали
- займ дали, банкрот
- займ дали, не банкрот

- Планируется построить две модели:

- анализ выдачи займа (таргет: выдан займ)
- анализ банкротства (таргет: банкрот)

Формирование тестовой выборки будет производится с учетом соотношения этих групп

```
Ввод [76]: def split_data(df, test_val_size=0.2, test_size=0.5):
    """ Функция разделения данных с учетом группировки данных на два таргета (дать займ?/человек банкрот?)
        :df - исходный датафрейм
        :test_val_size - размер тестовой+валидационной выборки относительно всех данных
        :test_size - доля разделения между валидационными и тестовыми данными

    :return - train_data, test_data, val_data - тренировочная/тестовая/валидационная выборки
    """
    # Разделяем данные на три группы:
    # 1. займ не дали
    group_data_1 = df[df["accept"] == 0]
    # 2. займ дали, банкрот
    group_data_2 = df[(df["accept"] == 1)&(df["target"] == 1)]
    # 3. займ дали, не банкрот
    group_data_3 = df[(df["accept"] == 1)&(df["target"] == 0)]
    assert df.shape[0] == (group_data_1.shape[0] + group_data_2.shape[0] + group_data_3.shape[0]), "При разделении на группы потеряны данные"

    # Формируем трайн/тест данные на основе трех групп
    # 1. займ не дали
    # Формируем train/test/val
    if len(group_data_1) == 0:
        train_group_data_1 = test_group_data_1 = val_group_data_1 = pd.DataFrame()
    else:
        train_group_data_1, test_group_data_1 = train_test_split(group_data_1, test_size=test_val_size, shuffle=True, random_state=53)
        test_group_data_1, val_group_data_1 = train_test_split(test_group_data_1, test_size=test_size, shuffle=True, random_state=53)
    # 2. займ дали, банкрот
    train_group_data_2, test_group_data_2 = train_test_split(group_data_2, test_size=test_val_size, shuffle=True, random_state=53)
    test_group_data_2, val_group_data_2 = train_test_split(test_group_data_2, test_size=test_size, shuffle=True, random_state=53)
    # 3. займ дали, не банкрот
    train_group_data_3, test_group_data_3 = train_test_split(group_data_3, test_size=test_val_size, shuffle=True, random_state=53)
    test_group_data_3, val_group_data_3 = train_test_split(test_group_data_3, test_size=test_size, shuffle=True, random_state=53)

    train_data = pd.concat([train_group_data_1, train_group_data_2, train_group_data_3])
    test_data = pd.concat([test_group_data_1, test_group_data_2, test_group_data_3])
    val_data = pd.concat([val_group_data_1, val_group_data_2, val_group_data_3])

    assert df.shape[0] == (train_data.shape[0] + test_data.shape[0] + val_data.shape[0]), "При разделении на тест/трайн группы потеряны данные"
    return train_data, test_data, val_data
```

```
Ввод [77]: train_data, test_data, val_data = split_data(train_df)
train_data.shape, test_data.shape, val_data.shape
```

```
Out[77]: ((118415, 72), (14802, 72), (14803, 72))
```

```

Ввод [78]: # Проверяем распределение банкротов и факта выдачи займа (для train/test/val)

# выданы займы
count_accept_all = train_df[train_df['accept'] == 0].shape[0]
count_accept_train = train_data[train_data['accept'] == 0].shape[0]
count_accept_test = test_data[test_data['accept'] == 0].shape[0]
count_accept_val = val_data[val_data['accept'] == 0].shape[0]

print(f"Займ дали/не дали: {count_accept_all/len(train_df)}")
assert int(count_accept_all/len(train_df)*100) == int(count_accept_train/len(train_data)*100)
assert int(count_accept_all/len(train_df)*100) == int(count_accept_test/len(test_data)*100)
assert int(count_accept_all/len(train_df)*100) == int(count_accept_val/len(val_data)*100)

# банкроты среди всех
count_bankrupt_all = train_df[train_df['target'] == 1].shape[0]
count_bankrupt_train = train_data[train_data['target'] == 1].shape[0]
count_bankrupt_test = test_data[test_data['target'] == 1].shape[0]
count_bankrupt_val = val_data[val_data['target'] == 1].shape[0]

print(f"Банкрот/не банкрот (для всей выборки): {count_bankrupt_all/len(train_df)}")
assert int(count_bankrupt_all/len(train_df)*100) == int(count_bankrupt_train/len(train_data)*100)
assert int(count_bankrupt_all/len(train_df)*100) == int(count_bankrupt_test/len(test_data)*100)
assert int(count_bankrupt_all/len(train_df)*100) == int(count_bankrupt_val/len(val_data)*100)

# банкроты среди выданных займов
count_bankrupt_accept_all = train_df[(train_df['accept'] == 1)&(train_df['target'] == 1)].shape[0]
count_bankrupt_accept_train = train_data[(train_data['accept'] == 1)&(train_data['target'] == 1)].shape[0]
count_bankrupt_accept_test = test_data[(test_data['accept'] == 1)&(test_data['target'] == 1)].shape[0]
count_bankrupt_accept_val = val_data[(val_data['accept'] == 1)&(val_data['target'] == 1)].shape[0]

print(f"Банкрот/не банкрот (для тех кому дали займ): {count_bankrupt_accept_all/len(train_df[(train_df['accept'] == 1)])}")
assert int(count_bankrupt_accept_all/len(train_df[(train_df['accept'] == 1)])*100) == int(count_bankrupt_accept_train/len(train_data[(train_data['accept'] == 1)])*100)
assert int(count_bankrupt_accept_all/len(train_df[(train_df['accept'] == 1)])*100) == int(count_bankrupt_accept_test/len(test_data[(test_data['accept'] == 1)])*100)
assert int(count_bankrupt_accept_all/len(train_df[(train_df['accept'] == 1)])*100) == int(count_bankrupt_accept_val/len(val_data[(val_data['accept'] == 1)])*100)

Займ дали/не дали: 0.714221051209296
Банкрот/не банкрот (для всей выборки): 0.07913795433049588
Банкрот/не банкрот (для тех кому дали займ): 0.27692016737192976

```

```

Ввод [79]: # Обновляем значения feature_columns после предварительной обработки
target_columns = ["target", "accept"]
feature_columns = train_df.columns.drop(target_columns)

```

Baseline модель

Условие задачи: "Построить базовую модель прогнозирования банкротства, одобряющую не менее 35% клиентов при банкротстве среди одобренных не выше 15%"
Т.к. единственное требование к одобренным клиентам, это чтобы они не были банкротами, тогда по условию требуется построить одну модель для прогнозирования: будет ли человек банкротом или нет.

Однако имея данные о выдаче/невыдаче займа. Можно построить вторую модель и результаты её прогноза подавать как признак в модель банкротства. Чтобы избежать лика данных надо аккуратно использовать стекинг (блэндинг на k-фолдах). Данный подход можно рассматривать как дальнейшее улучшение.

Также вторая модель позволит лучше понять природу данных и их взаимосвязь с правилами принятия решения.

План:

1. Построим обе модели:
 - прогнозирование: давать займ или нет
 - прогнозирование: будет ли человек банкротом или нет
2. По модели банкротство посчитаем искомые метрики (по ТЗ) "Построить базовую модель прогнозирования банкротства, одобряющую не менее 35% клиентов при банкротстве среди одобренных не выше 15%"
3. Анализ признаков, влияющих на предсказание. Для построения бейзлайн модели признаки не оптимизировались. Однако для итоговой модели потребуется провести работу по анализу признакового пространства.
4. Подбор гиперпараметров

```

Ввод [80]: # 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 datafram
    """
    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 [81]: # Выход 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=f'ROC curve (area = {roc_auc:.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
```

Baseline модель выдачи займа

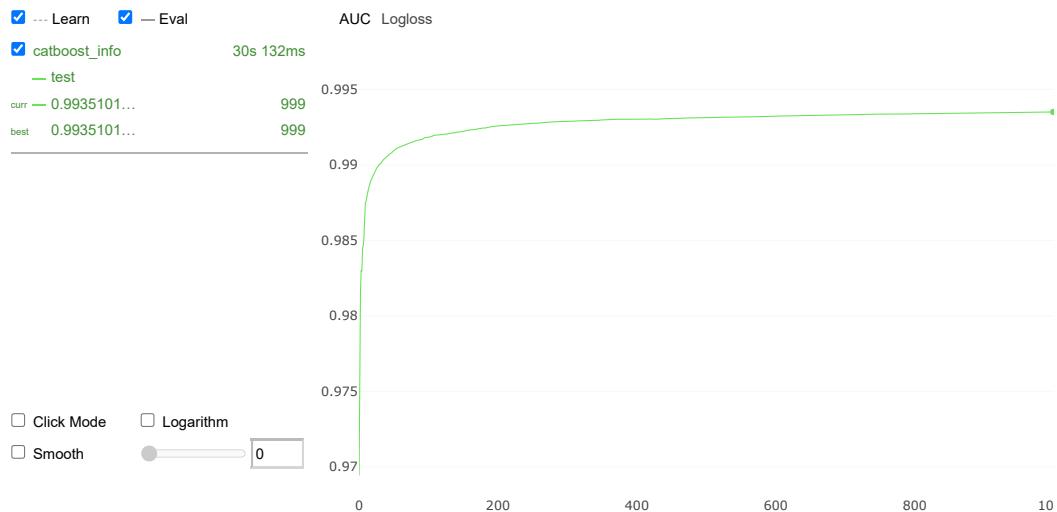
```
Ввод [82]: # Модель выдачи займа
accept_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=200)

X_train_accept = train_data[feature_columns]
y_train_accept = train_data['accept']

X_val_accept = val_data[feature_columns]
y_val_accept = val_data['accept']

X_test_accept = test_data[feature_columns]
y_test_accept = test_data['accept']
```

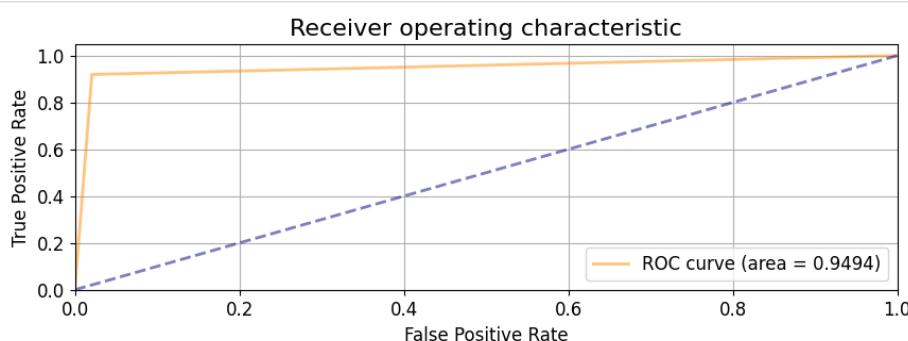
```
Ввод [83]: accept_model.fit(X_train_accept, y_train_accept, eval_set=(X_val_accept, y_val_accept), plot=True, verbose=False)
```



```
Out[83]: <catboost.core.CatBoostClassifier at 0x13199c01dc8>
```

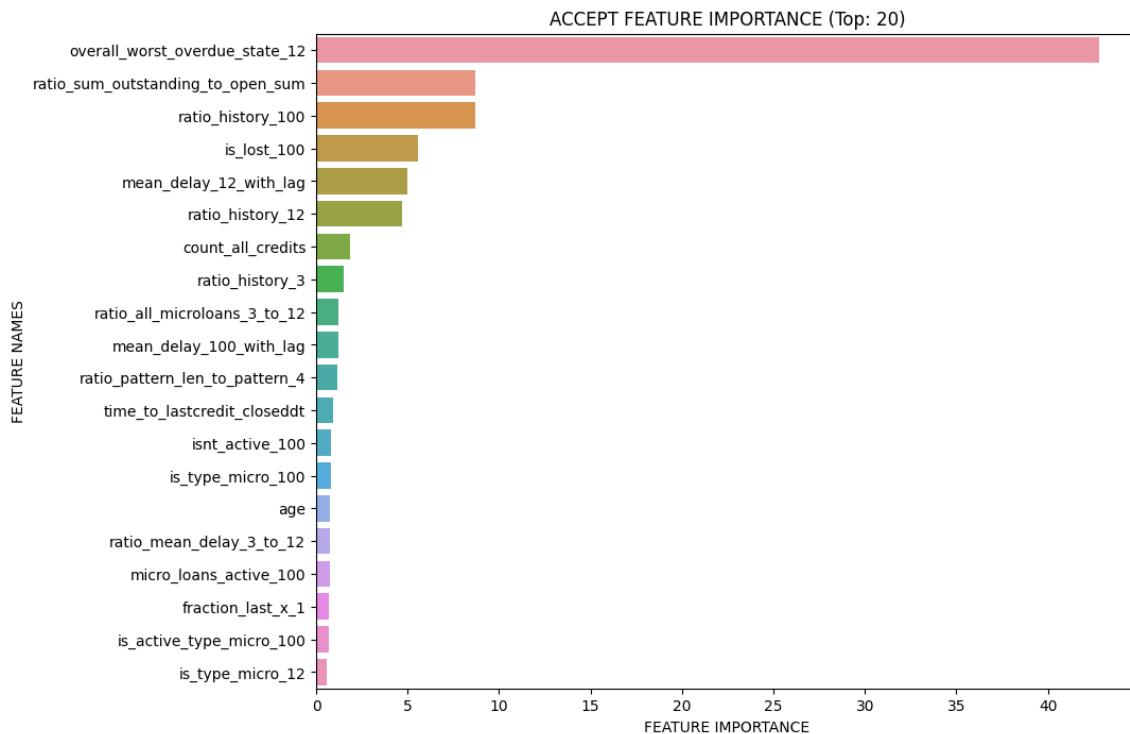
```
Ввод [84]: # Для расчета ROC-AUC на baseline модели используем тестовые данные
y_pred_accept = accept_model.predict(X_test_accept)
```

```
Ввод [85]: # Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_accept, y_pred=y_pred_accept)
roc_auc
```



```
Out[85]: 0.9493824178949882
```

```
Ввод [86]: # Важность признаков
fi_df = plot_feature_importance(accept_model.get_feature_importance(), X_test_accept.columns, model_name='ACCEPT', top_n=20)
```



```
Ввод [87]: # Feature importance в виде таблицы
fi_df
```

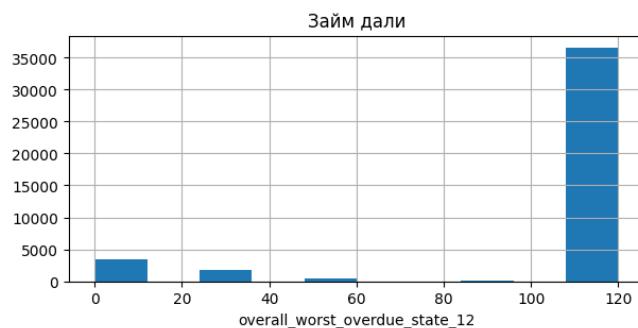
Out[87]:

	feature_names	feature_importance
68	overall_worst_overdue_state_12	42.813026
69	ratio_sum_outstanding_to_open_sum	8.697324
25	ratio_history_100	8.694904
7	is_lost_100	5.580312
33	mean_delay_12_with_lag	4.996206
...
59	is_active_type_credit_card_3	0.003900
66	is_active_type_consumer_1	0.003356
40	ratio_pattern_len_to_pattern_3	0.002682
62	is_type_credit_card_1	0.000000
17	is_lost_3	0.000000

70 rows × 2 columns

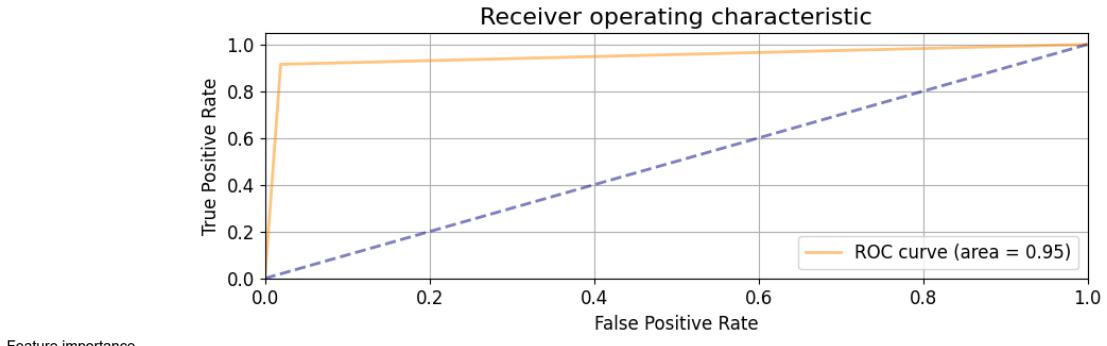
```
Ввод [88]: # Проверим распределение "overall_worst_overdue_state_12" на каждой группе таргетом
```

```
# 1. Займ дали
plt.figure(figsize=(7,3))
plt.title("Займ дали")
plt.xlabel("overall_worst_overdue_state_12")
train_df[train_df["accept"] == 1]["overall_worst_overdue_state_12"].hist()
plt.show()
# 2. Займ не дали
plt.figure(figsize=(7,3))
plt.title("Займ не дали")
plt.xlabel("overall_worst_overdue_state_12")
train_df[train_df["accept"] == 0]["overall_worst_overdue_state_12"].hist()
plt.show()
```

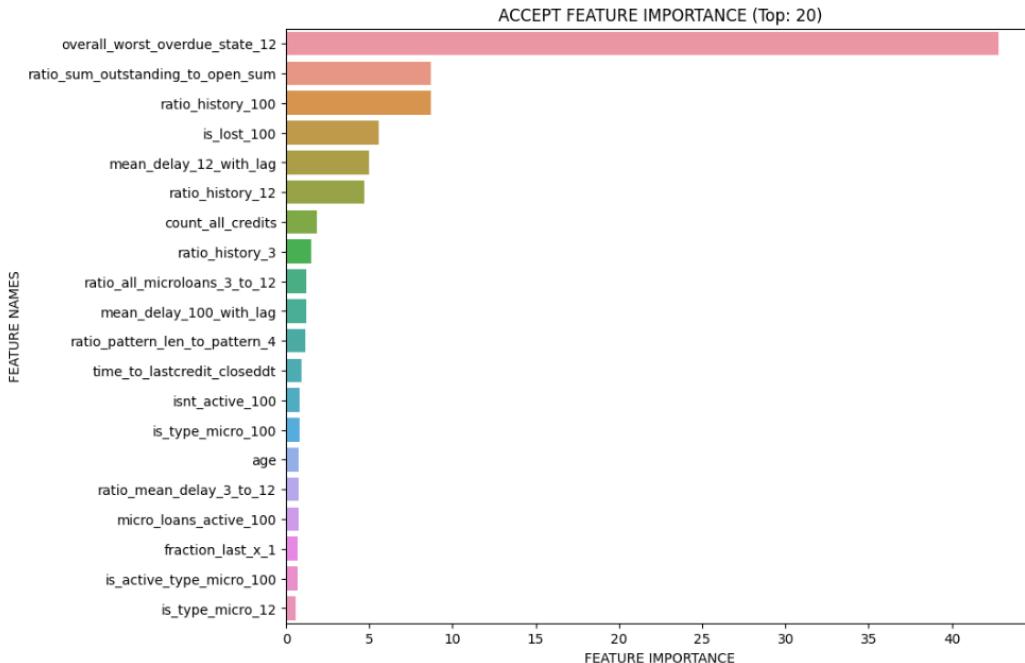


Предварительный анализ модели выдачи займа

Судя по кривой ROC-AUC, модель очень хорошо определяет кому выдать заем. Проблема в том что судя по кривой она слишком хорошо определяет это (ROC-AUC = 0.95). Анализируя важность признаков, наблюдаем, что признак "`overall_worst_overdue_state_12`" является решающим для прогнозирования выдачи займа (при анализе корреляции это признак также был самым значимым). Для признака "`overall_worst_overdue_state_12`" нет описания, но судя по названию, это общий наихудший статус по просроченным платежам за



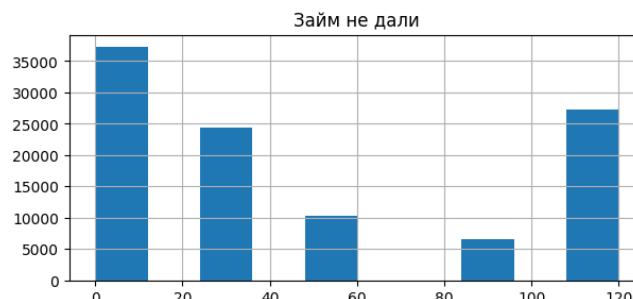
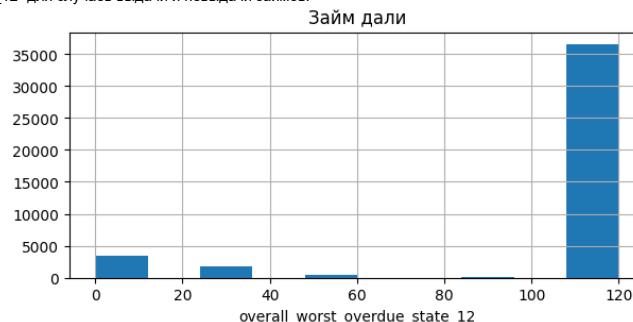
Feature importance



Учитывая, что признак "overall_worst_overdue_state_12" на порядки сильнее влияет на ответ (но по нему нет описания), можно сделать следующие предположения:

- Признак "overall_worst_overdue_state_12" содержит дополнительную информацию, которая не участвовала в остальных признаках. Т.е. этот признак содержит такую информацию которая не использовалась в других агрегационных признаках (например это статус благонадежности человека из внешней базы).
- Это действительно очень удачный признак, использующий известные данные. Например этот признак мог быть сгенерирован другой моделью машинного обучения.
- Этот признак является решающим для принятия решения о выдаче займа. По условию: отказ в займе может быть по причине отказа финансовой организации или нежеланию самого клиента. Если допустить, что людей которые подали заявку на займ, а потом передумали меньшество, тогда большая доля отказов по причине отказа финансовой организации. Для отказов действует набор правил по которым происходит отказ клиентам, и в таком случае "overall_worst_overdue_state_12" является решающим правилом (наи更重要ым).
- При формировании признака мог быть допущен лик данных (утечка таргета). Тогда он очень хорошо разделяет известные данные, но на неизвестных результат уже не будет таким же.

Распределение "overall_worst_overdue_state_12" для случаев выдачи и невыдачи займов:



Baseline модель определение банкротства

Обучение модели осуществляется только на части данных, на тех людях которым был выдан кредит, только по ним есть информация банкрот или нет.
Необходимо учитывать дисбаланс классов, используем для этого compute_class_weight от sklearn

Расчет метрик по условию задачи:

Исходное условие: "построить базовую модель прогнозирования банкротства, одобряющую не менее 35% клиентов при банкротстве среди одобренных не выше 15%."

Стандартная метрика recall не подойдет, потому что она считает полноту от общего объема данных, а нам необходимо посчитать банкротов относительно только одобренных клиентов.

В итоге нужны две метрики:

1. Доля клиентов которым модель одобрила кредит (т.е. сколько клиентов не являются банкротами по прогнозу модели) - должно быть не менее 35%
2. Доля клиентов которым модель одобрила кредит, но они являются банкротами - не более 15%

С помощью порога (threshold) можем регулировать категоричность (уверенность) модели в признании банкротства

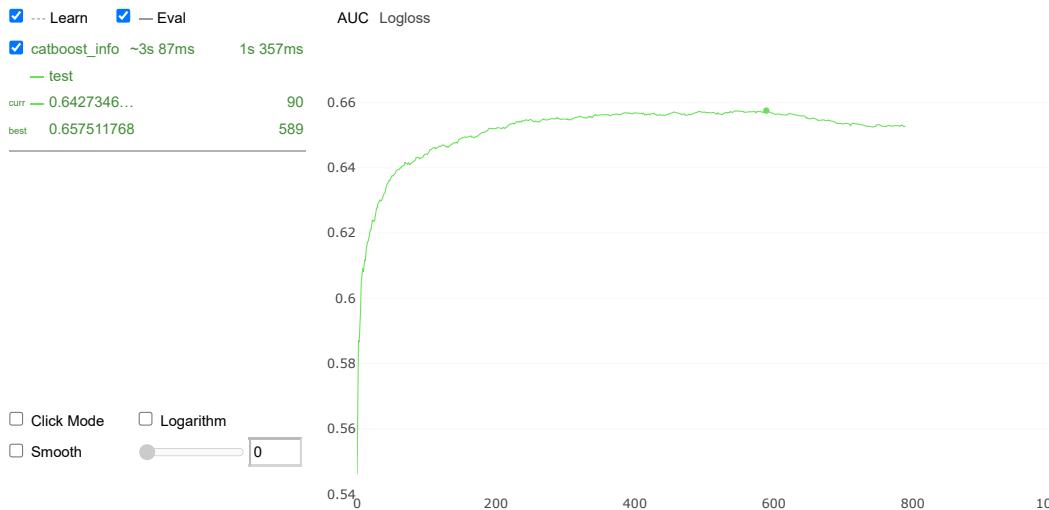
```
Ввод [89]: def calc_metrics(y_true, y_pred):  
    """Расчет метрик: скольким людям одобрен кредит и сколько из них банкроты  
    :y_true - реальный таргет банкротства  
    :y_pred - прогнозный таргет банкротства  
  
    return  
        :metric_clients_accept - доля клиентов которым модель одобрила кредит (т.к. не являются банкротами, по прогнозам модели)  
        :metric_clients_bankrupt - доля клиентов которым модель одобрила кредит, но они являются банкротами  
    """  
    pred_df = pd.DataFrame({"y_true":y_true, "y_pred":y_pred})  
    # Общее кол-во клиентов  
    clients = len(y_pred)  
    if clients == 0: return 0, 0  
    # Кол-во клиентов кому одобрили кредит - это те клиенты ,которым модель предсказали "не банкрот"  
    clients_accept = len(pred_df[pred_df["y_pred"] == 0])  
    # Кол-во клиентов банкроты из одобривших. Это те люди, где модель сказала, что человек "не банкрот", но в итоге ошиблась  
    clients_bankrupt = len(pred_df[(pred_df["y_pred"] == 0) & (pred_df["y_true"] == 1)])  
  
    # Доля клиентов которым модель одобрила кредит (т.к. не являются банкротами по прогнозу модели)  
    metric_clients_accept = clients_accept / clients  
    # Доля клиентов которым модель одобрила кредит, но они являются банкротами  
    metric_clients_bankrupt = clients_bankrupt / clients_accept  
    print(f"Одобрено {int(metric_clients_accept*100)}% клиентов")  
    print(f"Среди одобривших {int(metric_clients_bankrupt*100)}% клиентов-банкротов")  
    return metric_clients_accept, metric_clients_bankrupt
```

```
Ввод [90]: # Определяем подвыборку данных, рассматриваем только тех кому был выдан кредит  
X_train_bankrupt = train_data[train_data['accept'] == 1]  
y_train_bankrupt = X_train_bankrupt['target']  
X_train_bankrupt = X_train_bankrupt[feature_columns]  
  
X_val_bankrupt = val_data[val_data['accept'] == 1]  
y_val_bankrupt = X_val_bankrupt['target']  
X_val_bankrupt = X_val_bankrupt[feature_columns]  
  
X_test_bankrupt = test_data[test_data['accept'] == 1]  
y_test_bankrupt = X_test_bankrupt['target']  
X_test_bankrupt = X_test_bankrupt[feature_columns]  
  
X_train_bankrupt.shape, y_train_bankrupt.shape, X_val_bankrupt.shape, y_val_bankrupt.shape, X_test_bankrupt.shape, y_test_bankrupt.shape
```

Out[90]: ((33840, 70), (33840,), (4231, 70), (4231,), (4230, 70), (4230,))

```
Ввод [91]: # Модель определения банкрота  
bankrupt_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=200)
```

```
Ввод [92]: bankrupt_model.fit(X_train_bankrupt, y_train_bankrupt, eval_set=(X_val_bankrupt, y_val_bankrupt), plot=True, verbose=False)
```



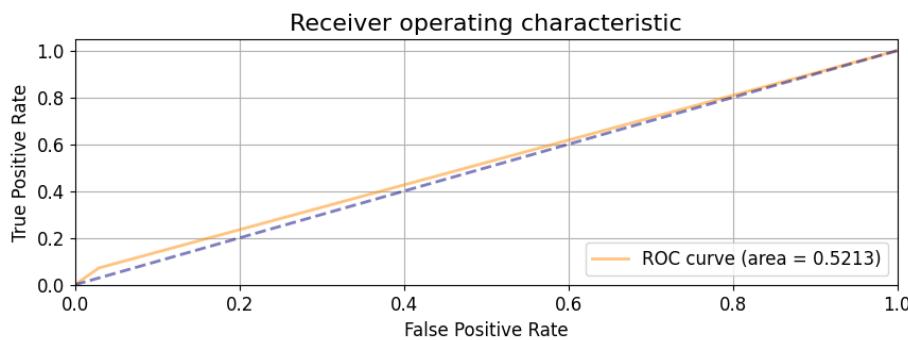
Out[92]: <catboost.core.CatBoostClassifier at 0x13198c265c8>

```
Ввод [93]: # Делаем предсказание на тестовых данных  
y_pred_bankrupt = bankrupt_model.predict(X_test_bankrupt)
```

```
Ввод [94]: # Строим график ROC-AUC
```

```
roc_auc = plot_roc_auc(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
```

```
roc_auc
```

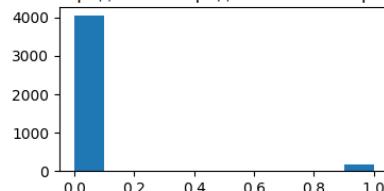


```
Out[94]: 0.5213195428700962
```

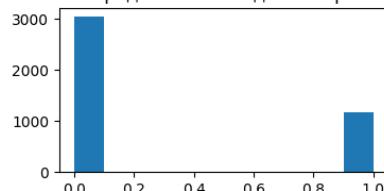
```
Ввод [95]: # Проверяем распределение исходного таргета на тестовых данных и предсказанного
```

```
plt.figure(figsize=(4,2))
plt.hist(y_pred_bankrupt)
plt.title("Распределение предсказанного таргета")
plt.figure(figsize=(4,2))
plt.hist(y_test_bankrupt)
plt.title("Распределение исходного таргета")
plt.show()
print("Распределение предсказанного таргета: {sum(y_pred_bankrupt)/len(y_pred_bankrupt)}")
print("Распределение исходного таргета: {sum(y_test_bankrupt)/len(y_test_bankrupt)}")
```

Распределение предсказанного таргета



Распределение исходного таргета



```
Распределение предсказанного таргета: 0.0408983451536643
```

```
Распределение исходного таргета: 0.2768321513002364
```

```
Ввод [96]: metric_clients_accept, metric_clients_bankrupt = calc_metrics(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
```

```
Одобрено 95% клиентов
```

```
Среди одобренных 26% клиентов-банкротов
```

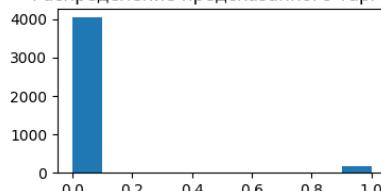
```
Out[96]: (0.9591016548463357, 0.2679319694355435)
```

Результат работы базовой модели:

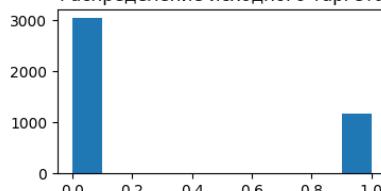
- ROC-AUC: 0.5213
- Распределение предсказанного таргета: 0.04
- Распределение исходного таргета: 0.28
- Одобрено 95% клиентов
- Среди одобренных 26% клиентов-банкротов

Легко заметить, что модель плохо выделяет банкротов, преобладает 0 значение таргета, если в тестовых данных распределение 0.28, то в предсказании модели это распределение уже 0.04:

Распределение предсказанного таргета



Распределение исходного таргета



Т.е. модель большинству людей одобряет заем (95%), но среди них попадает большое кол-во банкротов (26%).

Результаты экспериментов будем записывать в таблицу:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Распределение предсказаний	Одобрено клиентов	% банкротов
1	Базовая модель	Простая модель	0.6575	0.5213	0.04	95%	26%

Предварительный план улучшения модели

Варианты улучшения:

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

Добавляем балансировку классов

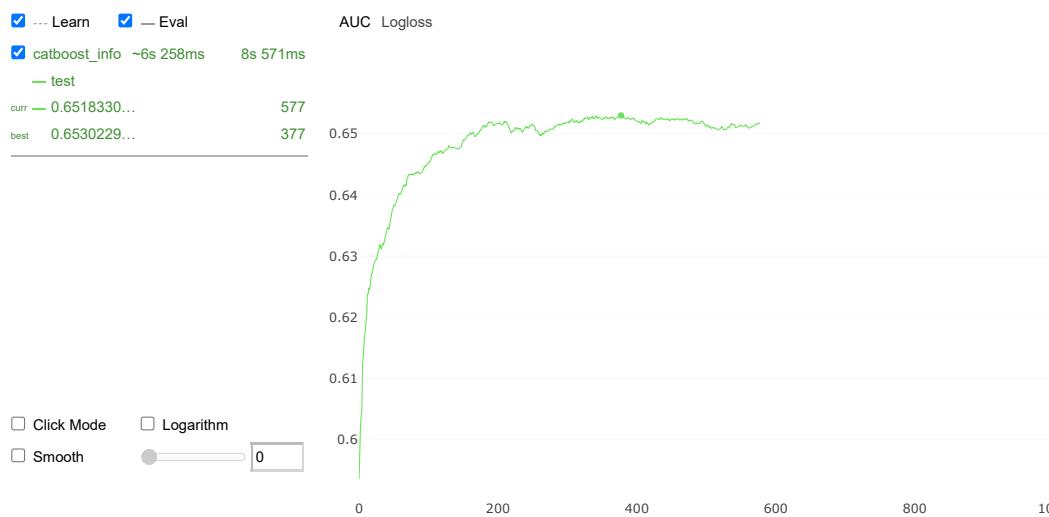
Ввод [97]:

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

Out[97]: {0.0: 0.6914871878703666, 1.0: 1.805570376694056}

Ввод [98]:

```
# Модель определения банкрота
bankrupt_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=200, class_weights=class_weights)
bankrupt_model.fit(X_train_bankrupt, y_train_bankrupt, eval_set=(X_val_bankrupt, y_val_bankrupt), plot=True, verbose=False)
```



Out[98]: <catboost.core.CatBoostClassifier at 0x13199410688>

Ввод [99]:

```
# Делаем предсказание на тестовых данных
y_pred_bankrupt = bankrupt_model.predict(X_test_bankrupt)
```

Ввод [100]:

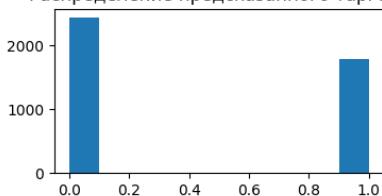
```
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
roc_auc
```



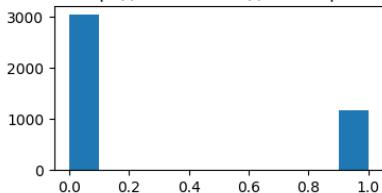
Out[100]: 0.6104260670240188

```
Ввод [101]: # Проверяем распределение исходного таргета на тестовых данных и предсказанного
plt.figure(figsize=(4,2))
plt.hist(y_pred_bankrupt)
plt.title("Распределение предсказанного таргета")
plt.figure(figsize=(4,2))
plt.hist(y_test_bankrupt)
plt.title("Распределение исходного таргета")
plt.show()
print(f"Распределение предсказанных таргетов: {sum(y_pred_bankrupt)/len(y_pred_bankrupt)}")
print(f"Распределение исходных таргетов: {sum(y_test_bankrupt)/len(y_test_bankrupt)}")
```

Распределение предсказанного таргета



Распределение исходного таргета



Распределение предсказанных таргетов: 0.4226950354609929
Распределение исходных таргетов: 0.2768321513002364

```
Ввод [102]: metric_clients_accept, metric_clients_bankrupt = calc_metrics(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
metric_clients_accept, metric_clients_bankrupt
```

Одобрено 57% клиентов
Среди одобренных 20% клиентов-банкротов

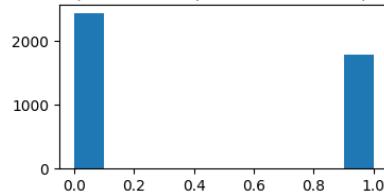
```
Out[102]: (0.577304964539007, 0.20024570024570024)
```

Результат работы модели с учетом дисбаланса классов:

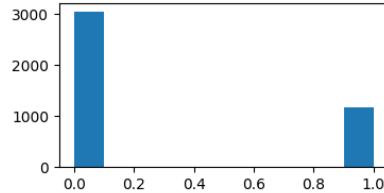
- ROC-AUC: 0.6104
- Распределение предсказанных таргетов: 0.42
- Распределение исходного таргета: 0.28
- Одобрено 57% клиентов
- Среди одобренных 20% клиентов-банкротов

После учета дисбаланса классов распределение предсказанных таргета изменилось с 0.04 до 0.42 - неплохо:

Распределение предсказанного таргета



Распределение исходного таргета



Управление дисбалансом можно вынести в отдельный гиперпараметр!

Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Распределение предсказаний	Одобрено клиентов	% банкротов
1	Базовая модель	Простая модель	0.6575	0.5213		0.04	95%
2	Модель №1 + дисбаланс классов	Учитываем в модели дисбаланс классов	0.6530	0.6104		0.42	57%

Выбор порога threshold

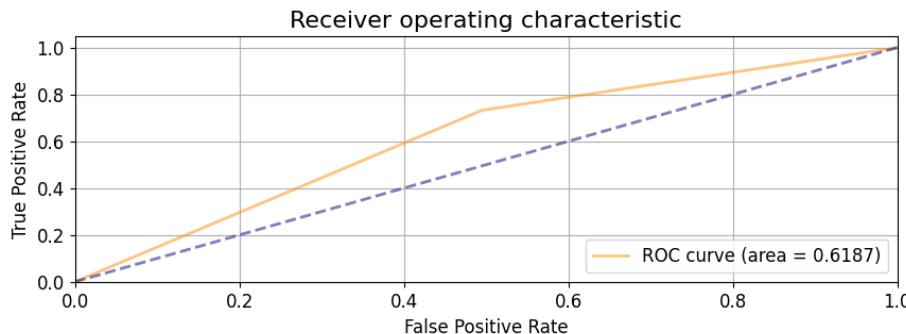
```
Ввод [103]: # Делаем предсказание модели (на тестовых данных) через predict_proba
y_pred_proba_bankrupt = bankrupt_model.predict_proba(X_test_bankrupt)
```

```
Ввод [104]: # Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test_bankrupt, y_score=y_pred_proba_bankrupt[:, 1])
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")
```

Оптимальный порог: 0.4562697047919134

```
Ввод [105]: # формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > best_thresh).astype(int)
```

```
Ввод [106]: # Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
roc_auc
```



```
Out[106]: 0.6187235995532216
```

```
Ввод [107]: print(f"Распределение предсказанного таргета: {sum(y_pred_bankrupt)/len(y_pred_bankrupt)}")
print(f"Распределение исходного таргета: {sum(y_test_bankrupt)/len(y_test_bankrupt)}")
```

Распределение предсказанного таргета: 0.5609929078014184
Распределение исходного таргета: 0.2768321513002364

```
Ввод [108]: metric_clients_accept, metric_clients_bankrupt = calc_metrics(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
metric_clients_accept, metric_clients_bankrupt
```

Одобрено 43% клиентов
Среди одобренных 16% клиентов-банкротов

```
Out[108]: (0.4390070921985816, 0.16855142703284867)
```

Результат работы модели с учетом оптимального порога (threshold)

- ROC-AUC: 0.6187
- Распределение предсказанного таргета: 0.56
- Распределение исходного таргета: 0.28
- Одобрено 43% клиентов
- Среди одобренных 16% клиентов-банкротов

Процент клиентов-банкротов среди одобренных снизился с 20% до 16%.

Выбор threshold можно вынести в отдельный гиперпараметр!

Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Распределение предсказаний	Одобрено клиентов	% банкротов
1	Базовая модель	Простая модель	0.6575	0.5213	0.04	95%	26%
2	Модель №1 + дисбаланс классов	Учитываем в модели дисбаланс классов	0.6530	0.6104	0.42	57%	20%
3	Модель №2 + оптимальный threshold	Определяем для модели оптимальный threshold	0.6530	0.6187	0.56	43%	16%

Feature engineering

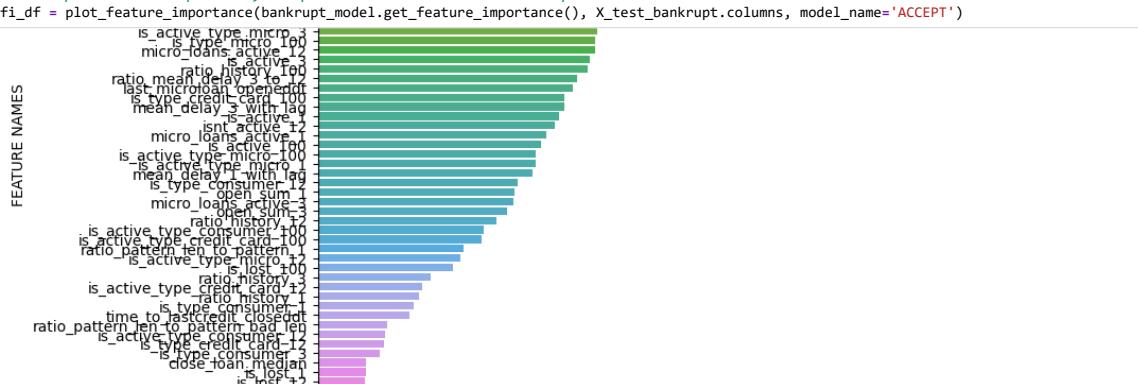
Большой блок задач это подбор признаков.

Разобьем на подзадачи:

1. Генерация новых признаков
2. Исключение признаков не влияющих на таргет

Генерация новых признаков

```
Ввод [109]: # Посмотрим на Топ 30 признаков, которые имеют влияние на модель банкротства
fi_df = plot_feature_importance(bankrupt_model.get_feature_importance(), X_test_bankrupt.columns, model_name='ACCEPT')
```



```
Ввод [110]: # не будем загрязнять существующие датасеты, сделаем копию
bankrupt_df = train_df[train_df["accept"] == 1].copy()
bankrupt_df.shape
```

```
Out[110]: (42301, 72)
```

```

Ввод [181]: %%time
# формируем матрицу корреляции
corr_matrix = bankrupt_df.corr()

Wall time: 550 ms

Ввод [182]: n = 25
# Берем 20 признаков с наибольшей корреляцией с таргетом
base_features = set(abs(corr_matrix["target"].drop(["accept", "target"], axis=0)).sort_values(ascending=False)[:n].index)
# Берем 20 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"].values)

e = 1e-5 # добавляем минимальное значение, чтобы уйти от деления на ноль и появления бесконечностей
# Сгенерируем новые признаки, которые будут представлять отношение одного к другому, например отношение общей суммы кредитов к общему кол-ву выданных и т.д.
for base_f_1 in base_features:
    for base_f_2 in base_features:
        bankrupt_df[f"new_{base_f_1}_{base_f_2}"] = (bankrupt_df[base_f_1] + e)/(bankrupt_df[base_f_2] + e)
# Сгенерируем дополнительные признаки относительно возраста и относительно статуса кредитоспособности ("overall_worst_overdue_state_12")
for base_f_1 in base_features:
    bankrupt_df[f"new_age_{base_f_1}"] = (bankrupt_df[base_f_1] + e) * bankrupt_df["age"]
    bankrupt_df[f"new_credit_state_{base_f_1}"] = (bankrupt_df[base_f_1] + e) * bankrupt_df["overall_worst_overdue_state_12"]
bankrupt_df.shape

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()`
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[182]: (42301, 1332)

Ввод [183]: # Заново переопределяем перечень фичей, но уже от custom_df
feature_columns_custom = bankrupt_df.columns.drop(target_columns)
feature_columns_custom.shape

Out[183]: (1330,)

Ввод [184]: train_data_custom, test_data_custom, val_data_custom = split_data(bankrupt_df)
train_data_custom.shape, test_data_custom.shape, val_data_custom.shape

Out[184]: ((33840, 1332), (4230, 1332), (4231, 1332))

Ввод [185]: # Определяем подвыборку данных, рассматриваем только тех кому был выдан кредит
X_train_bankrupt_custom = train_data_custom[train_data_custom['accept'] == 1]
y_train_bankrupt_custom = X_train_bankrupt_custom['target']
X_train_bankrupt_custom = X_train_bankrupt_custom[feature_columns_custom]

X_val_bankrupt_custom = val_data_custom[val_data_custom['accept'] == 1]
y_val_bankrupt_custom = X_val_bankrupt_custom['target']
X_val_bankrupt_custom = X_val_bankrupt_custom[feature_columns_custom]

X_test_bankrupt_custom = test_data_custom[test_data_custom['accept'] == 1]
y_test_bankrupt_custom = X_test_bankrupt_custom['target']
X_test_bankrupt_custom = X_test_bankrupt_custom[feature_columns_custom]

X_train_bankrupt_custom.shape, y_train_bankrupt_custom.shape, X_val_bankrupt_custom.shape, y_val_bankrupt_custom.shape, X_test_bankrupt_custom.shape, y_test_bankrupt_custom.shape

Out[185]: ((33840, 1330), (33840,), (4231, 1330), (4231,), (4230, 1330), (4230,))

Ввод [186]: # Модель определения банкрота
bankrupt_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=200, class_weights=class_weights, random_seed=53)
bankrupt_model.fit(X_train_bankrupt_custom, y_train_bankrupt_custom, eval_set=(X_val_bankrupt_custom, y_val_bankrupt_custom), plot=True, verbose=False)

... Learn   ✓ — Eval
          AUC Logloss
✓ catboost_info ~2m 26s      1m 30s
  — test
curr — 0.6569767...
best  0.6616153...
379
179
0.66
0.65
0.64
0.63
0.62
0
0  200  400  600  800  1000
Click Mode  Logarithm
Smooth  0

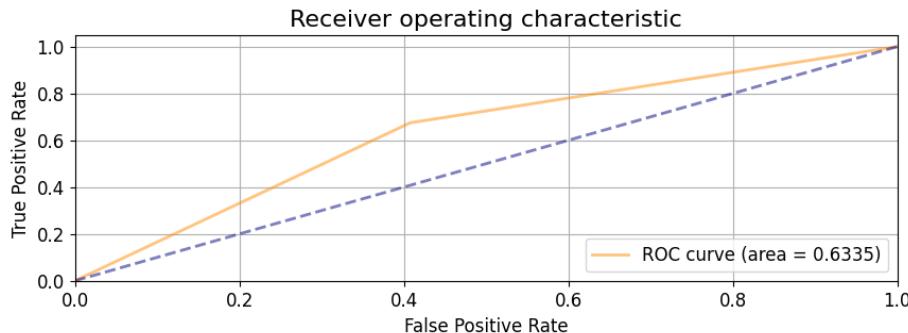
Out[186]: <catboost.core.CatBoostClassifier at 0x1319a022348>

Ввод [187]: # Делаем предсказание модели (на тестовых данных) через predict_proba
y_pred_proba_bankrupt = bankrupt_model.predict_proba(X_test_bankrupt_custom)

```

```
Ввод [193]: # Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test_bankrupt_custom,y_score=y_pred_proba_bankrupt[:, 1])
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")
# Формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > best_thresh).astype(int)
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_bankrupt_custom, y_pred=y_pred_bankrupt)
roc_auc
```

Оптимальный порог: 0.4862354095728443



Out[193]: 0.6334937518302868

```
Ввод [197]: # Формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > 0.435).astype(int)

print(f"Распределение предсказанного таргета: {sum(y_pred_bankrupt)/len(y_pred_bankrupt)}")
print(f"Распределение исходного таргета: {sum(y_test_bankrupt)/len(y_test_bankrupt)}")

metric_clients_accept, metric_clients_bankrupt = calc_metrics(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
metric_clients_accept, metric_clients_bankrupt
```

Распределение предсказанного таргета: 0.6212765957446809

Распределение исходного таргета: 0.2768321513002364

Одобрено 37% клиентов

Среди одобренных 14% клиентов-банкротов

Out[197]: (0.37872340425531914, 0.1441947565543071)

Результат работы модели с учетом генерации новых признаков

- ROC-AUC (val): 0.6616
- ROC-AUC (test): 0.6334
- Распределение предсказанного таргета: 0.62
- Распределение исходного таргета: 0.28
- Одобрено 37.8% клиентов
- Среди одобренных 14.4% клиентов-банкротов

Добавлены сгенерированные признаки. Однако новые признаки пока добавлены просто кучей, необходимо выбрать только действительно значимые среди них.

На данном этапе удалось выбрать threshold, с которым модель достигает требуемых показателей : Одобрено 35.8% клиентов, Среди одобренных 14.6% клиентов-банкротов.

Следующий этап выбор оптимальных признаков.

Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Распределение предсказаний	Одобрено клиентов	% банкротов
1	Базовая модель	Простая модель	0.6575	0.5213	0.04	95%	26%
2	Модель №1 + дисбаланс классов	Учитываем в модели дисбаланс классов	0.6530	0.6104	0.42	57%	20%
3	Модель №2 + оптимальный threshold	Определяем для модели оптимальный threshold	0.6530	0.6187	0.56	43%	16%
4	Модель №3 + новые признаки	Сгенерированы новые признаки и выбран threshold (0.435). Без чистки признаков. Достигнут требуемый скор!	0.6616	0.6334	0.62	37,8%	14.4%

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 [198]: %%time
custom_corr_matrix = bankrupt_df[list(X_val_bankrupt_custom.columns) + target_columns].corr()
custom_corr_matrix.shape
```

Wall time: 2min 59s

Out[198]: (1332, 1332)

```
Ввод [199]: # Оставляем признаки у которых есть корреляция с целевой переменной (таргетом)
target_corr = abs(custom_corr_matrix["target"]).sort_values(ascending=False)
# abs(custom_corr_matrix["target"]).sort_values(ascending=False)
features_target_corr = list(set(target_corr[target_corr > 0.01].index) - set(["target", "accept"]))
print(f"Выбрано {len(features_target_corr)} признаков")
```

Выбрано 937 признаков

```
Ввод [200]: # Убираем признаки с высоким уровнем корреляции (оставляем )
limit_corr_level = 0.85
select_features = []
skip_features = set([])

for feat in features_target_corr:
    if feat in skip_features:
        continue
    select_features.append(feat)
    skip_features |= set(custom_corr_matrix[abs(custom_corr_matrix[feat]) > limit_corr_level][feat].index)

print(f"Выбрано {len(select_features)} признаков")

Выбрано 365 признаков
```

```
Ввод [201]: # Посмотрим соотношение оригинальных признаков и сгенерированных
new_features = [feat for feat in select_features if "new" in feat]
print(f"Из общего числа выбранных признаков {len(select_features)}, новых {len(new_features)}")
```

Из общего числа выбранных признаков 365, новых 341

```
Ввод [202]: train_data_custom, test_data_custom, val_data_custom = split_data(bankrupt_df)
train_data_custom.shape, test_data_custom.shape, val_data_custom.shape
```

```
Out[202]: ((38340, 1332), (4230, 1332), (4231, 1332))
```

```
Ввод [203]: # Определяем подвыборку данных, рассматриваем только тех кому был выдан кредит
```

```
X_train_bankrupt_custom = train_data_custom[train_data_custom['accept'] == 1]
y_train_bankrupt_custom = X_train_bankrupt_custom['target']
X_train_bankrupt_custom = X_train_bankrupt_custom[select_features]

X_val_bankrupt_custom = val_data_custom[val_data_custom['accept'] == 1]
y_val_bankrupt_custom = X_val_bankrupt_custom['target']
X_val_bankrupt_custom = X_val_bankrupt_custom[select_features]

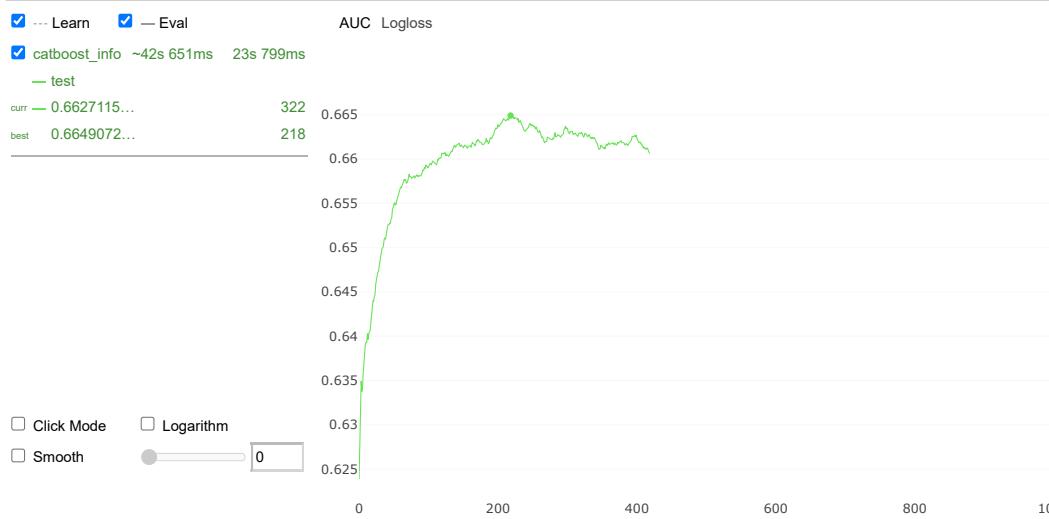
X_test_bankrupt_custom = test_data_custom[test_data_custom['accept'] == 1]
y_test_bankrupt_custom = X_test_bankrupt_custom['target']
X_test_bankrupt_custom = X_test_bankrupt_custom[select_features]
```

```
X_train_bankrupt_custom.shape, y_train_bankrupt_custom.shape, X_val_bankrupt_custom.shape, y_val_bankrupt_custom.shape, X_test_bankrupt_custom.shape, y_test_bankrupt_custom.shape
```

```
Out[203]: ((33840, 365), (33840,), (4231, 365), (4231,), (4230, 365), (4230,))
```

```
Ввод [204]: # Модель определения банкрота
```

```
bankrupt_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=200, class_weights=class_weights, random_seed=53)
bankrupt_model.fit(X_train_bankrupt_custom, y_train_bankrupt_custom, eval_set=(X_val_bankrupt_custom, y_val_bankrupt_custom), plot=True, verbose=False)
```



```
Out[204]: <catboost.core.CatBoostClassifier at 0x131b53e9188>
```

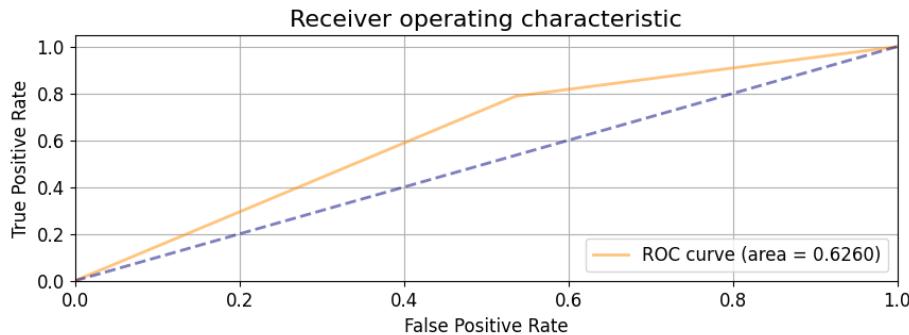
```
Ввод [205]: # Делаем предсказание модели (на тестовых данных) через predict_proba
y_pred_proba_bankrupt = bankrupt_model.predict_proba(X_test_bankrupt_custom)
```

```

Ввод [207]: # Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test_bankrupt_custom,y_score=y_pred_proba_bankrupt[:, 1])
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")
# Формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > best_thresh).astype(int)
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_bankrupt_custom, y_pred=y_pred_bankrupt)
roc_auc

```

Оптимальный порог: 0.43916992329197474



Out[207]: 0.6260461423487803

```

Ввод [217]: # Формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > 0.435).astype(int)

print(f"Распределение предсказанного таргета: {sum(y_pred_bankrupt)/len(y_pred_bankrupt)}")
print(f"Распределение исходного таргета: {sum(y_test_bankrupt)/len(y_test_bankrupt)}")

metric_clients_accept, metric_clients_bankrupt = calc_metrics(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
metric_clients_accept, metric_clients_bankrupt

```

Распределение предсказанного таргета: 0.6158392434988179
 Распределение исходного таргета: 0.2768321513002364
 Одобрено 38% клиентов
 Среди одобренных 14% клиентов-банкротов

Out[217]: (0.38416075650118203, 0.14892307692307694)

Результат работы модели с учетом выбора оптимальных признаков

Произведена чистка признаков. На данном этапе удалось выбрать threshold, с которым модель достигает требуемых показателей : Одобрено 38,4% клиентов, Среди одобренных 14,8% клиентов-банкротов.

Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Распределение предсказаний	Одобрено клиентов	% банкротов
1	Базовая модель	Простая модель	0.6575	0.5213	0.04	95%	26%
2	Модель №1 + дисбаланс классов	Учитываем в модели дисбаланс классов	0.6530	0.6104	0.42	57%	20%
3	Модель №2 + оптимальный threshold	Определяем для модели оптимальный threshold	0.6530	0.6187	0.56	43%	16%
4	Модель №3 + новые признаки	Сгенерированы новые признаки и выбран threshold (0.435). Без чистки признаков. Достигнут требуемый скор!	0.6616	0.6334	0.62	37,8%	14,4%
5	Модель №4 + чистка признаков	Убраны лишние признаки	0.6649	0.6260	0.61	38,4%	14,8%

Произведем кластеризацию данных

```

Ввод [137]: %time
# Долго строится ~5 мин
tsne_transform = TSNE(n_jobs=-1)
tsne_features = tsne_transform.fit_transform(bankrupt_df.drop(columns=["target", "accept"]))
scaler = StandardScaler()
tsne_features = scaler.fit_transform(tsne_features)
print(tsne_features.shape)

The default initialization in TSNE will change from 'random' to 'pca' in 1.2.
The default learning rate in TSNE will change from 200.0 to 'auto' in 1.2.

(42301, 2)
Wall time: 4min 58s

```

```

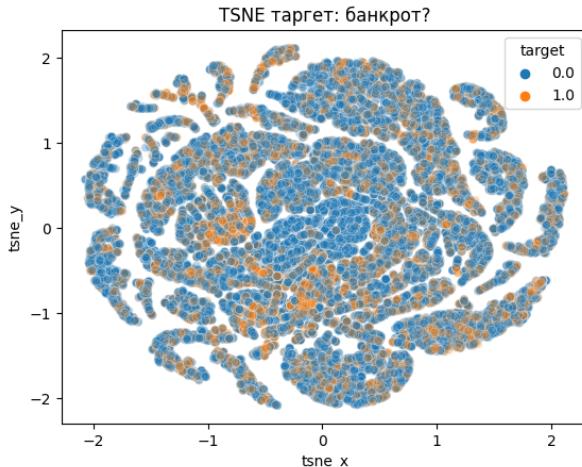
Ввод [138]: scatter_df = bankrupt_df[["target", "accept"]].copy()
scatter_df['tsne_x'] = tsne_features[:, 0]
scatter_df['tsne_y'] = tsne_features[:, 1]
scatter_df.shape

```

Out[138]: (42301, 4)

```
Ввод [139]: sns.scatterplot(x="tsne_x", y="tsne_y", hue="target", data=scatter_df, alpha=0.3)
plt.title("TSNE таргет: банкрот?")
plt.show
```

```
Out[139]: <function matplotlib.pyplot.show(close=None, block=None)>
```



Итог кластеризации: Первичная кластеризация показывает, что явного визуального разделения на классы не наблюдается. Однако есть подобласти которые например принадлежат полностью классу "0". Також есть подобласти где класс "1" преобладает. Т.е. некоторые закономерности возможны.

Сэмплиинг данных

В качестве борьбы с дисбалансом классов допускается использовать сэмплирование данных:

- исключение "выбросов" или простое случайное исключение из мажорного класса (Over Sampling)
- генерация схожих в минорном классе (Under Sampling)

Проверим как это будет применимо к нашим данным

```
Ввод [218]: print(f"Распределение классов: {y_train_bankrupt_custom.value_counts()}")
X_train_bankrupt_custom.shape, y_train_bankrupt_custom.shape
```

```
Распределение классов: 0.0    24469
1.0     9371
Name: target, dtype: int64
```

```
Out[218]: ((33840, 365), (33840,))
```

Пробуем уменьшить выборку мажорного класса к минорному. Случайной подвыборкой

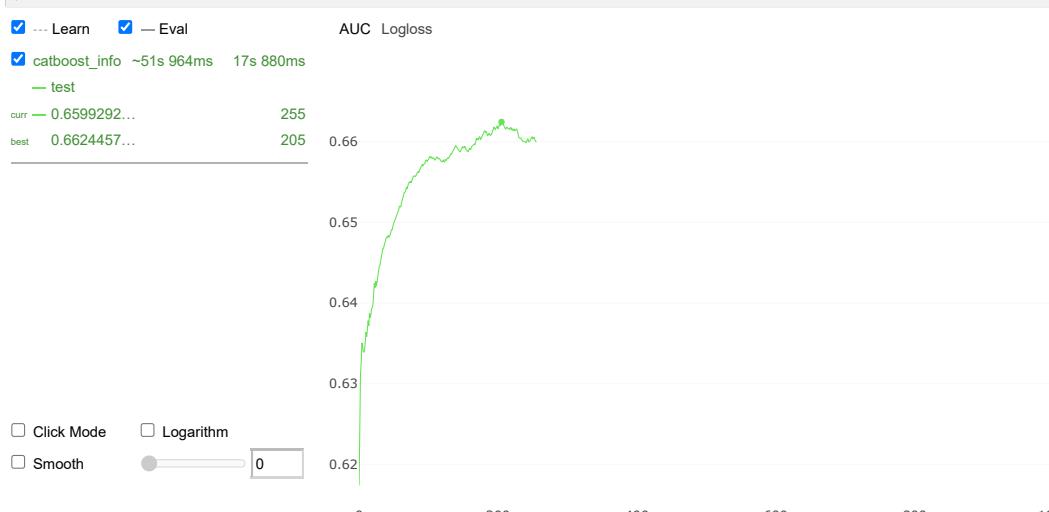
```
Ввод [219]: X_train_sampling, y_train_sampling = RandomUnderSampler().fit_resample(X_train_bankrupt_custom, y_train_bankrupt_custom)
print(f"Распределение классов: {y_train_sampling.value_counts()}")
X_train_sampling.shape, y_train_sampling.shape
```

```
Распределение классов: 0.0    9371
1.0     9371
Name: target, dtype: int64
```

```
Out[219]: ((18742, 365), (18742,))
```

```
Ввод [220]: # Перерасчет дисбаланса классов
classes = np.unique(y_train_sampling)
weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train_sampling)
sampling_class_weights = dict(zip(classes, weights))
sampling_class_weights

# Модель определения банкрота
bankrupt_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=50, class_weights=sampling_class_weights,)
bankrupt_model.fit(X_train_sampling, y_train_sampling, eval_set=(X_val_bankrupt_custom, y_val_bankrupt_custom), plot=True, verbose=False)
```



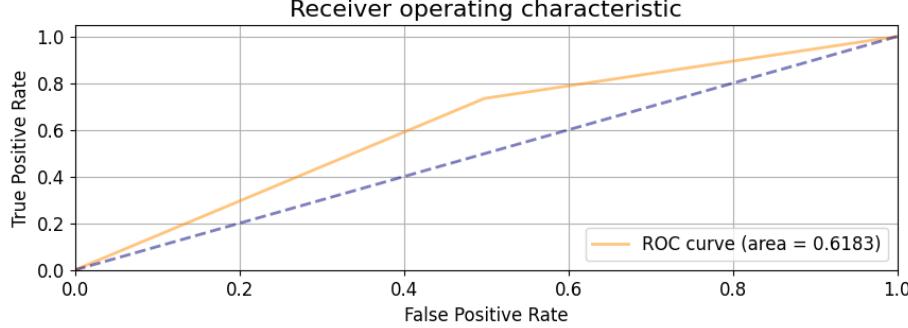
```
Out[220]: <catboost.core.CatBoostClassifier at 0x13198e76d08>
```

```

Ввод [221]: # Делаем предсказание модели (на тестовых данных) через predict_proba
y_pred_proba_bankrupt = bankrupt_model.predict_proba(X_test_bankrupt_custom)
# Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test_bankrupt_custom, y_score=y_pred_proba_bankrupt[:, 1])
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")
# Формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > best_thresh).astype(int)
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_bankrupt_custom, y_pred=y_pred_bankrupt)
roc_auc

```

Оптимальный порог: 0.4650804409248747



Out[221]: 0.6182699536499512

```

Ввод [222]: # формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > 0.425).astype(int)

print(f"Распределение предсказанного таргета: {sum(y_pred_bankrupt)/len(y_pred_bankrupt)}")
print(f"Распределение исходного таргета: {sum(y_test_bankrupt)/len(y_test_bankrupt)}")

metric_clients_accept, metric_clients_bankrupt = calc_metrics(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
metric_clients_accept, metric_clients_bankrupt

Распределение предсказанного таргета: 0.6683215130023641
Распределение исходного таргета: 0.2768321513002364
Одобрено 33% клиентов
Среди одобренных 14% клиентов-банкротов

```

Out[222]: (0.3316784869976359, 0.1482537419814683)

Пробуем выкинуть выбросы из мажорного класса

```

Ввод [223]: X_train_sampling, y_train_sampling = TomekLinks().fit_resample(X_train_bankrupt_custom, y_train_bankrupt_custom)
print(f"Распределение классов: {y_train_sampling.value_counts()}")
X_train_sampling.shape, y_train_sampling.shape

Распределение классов: 0.0    20832
1.0     9371
Name: target, dtype: int64

```

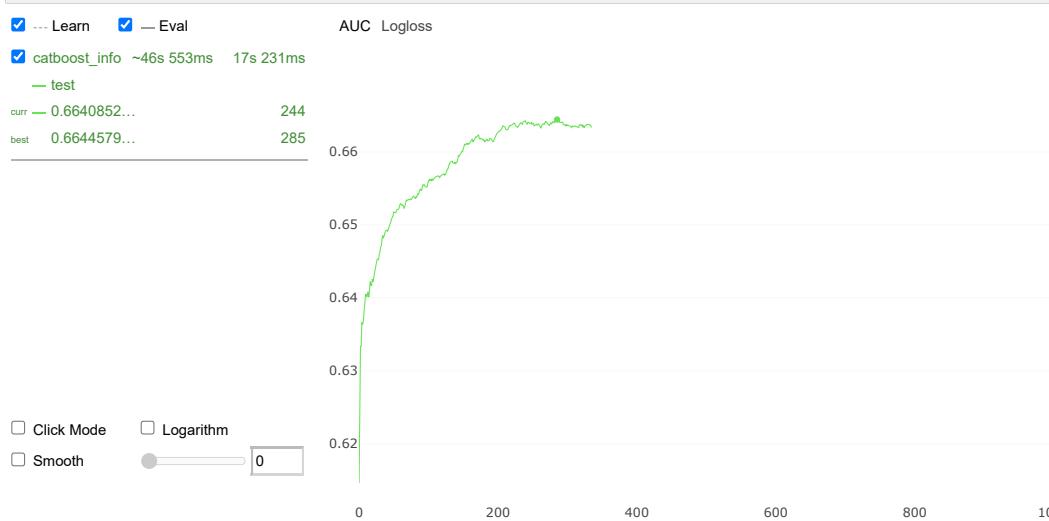
Out[223]: ((30203, 365), (30203,))

```

Ввод [224]: # Перерасчет дисбаланса классов
classes = np.unique(y_train_sampling)
weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train_sampling)
sampling_class_weights = dict(zip(classes, weights))
sampling_class_weights

# Модель определения банкрота
bankrupt_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=50, class_weights=sampling_class_weights,)
bankrupt_model.fit(X_train_sampling, y_train_sampling, eval_set=(X_val_bankrupt_custom, y_val_bankrupt_custom), plot=True, verbose=False)

```



Out[224]: <catboost.core.CatBoostClassifier at 0x131b4ef8c88>

```

Ввод [225]: # Делаем предсказание модели (на тестовых данных) через predict_proba
y_pred_proba_bankrupt = bankrupt_model.predict_proba(X_test_bankrupt_custom)
# Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test_bankrupt_custom, y_score=y_pred_proba_bankrupt[:, 1])
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")
# Формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > best_thresh).astype(int)
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_bankrupt_custom, y_pred=y_pred_bankrupt)
roc_auc

```

Оптимальный порог: 0.4475505616265961



Out[225]: 0.6250383505267457

```

Ввод [237]: # формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > 0.419).astype(int)

print(f"Распределение предсказанного таргета: {sum(y_pred_bankrupt)/len(y_pred_bankrupt)}")
print(f"Распределение исходного таргета: {sum(y_test_bankrupt)/len(y_test_bankrupt)}")

metric_clients_accept, metric_clients_bankrupt = calc_metrics(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
metric_clients_accept, metric_clients_bankrupt

Распределение предсказанного таргета: 0.6437352245862884
Распределение исходного таргета: 0.2768321513002364
Одобрено 35% клиентов
Среди одобренных 14% клиентов-банкротов

```

Out[237]: (0.3562647754137116, 0.1493032514930325)

Пробуем Over Sampler для минорного класса

```

Ввод [227]: X_train_sampling, y_train_sampling = RandomOverSampler().fit_resample(X_train_bankrupt_custom, y_train_bankrupt_custom)
print(f"Распределение классов: {y_train_sampling.value_counts()}")
X_train_sampling.shape, y_train_sampling.shape

Распределение классов: 1.0    24469
0.0    24469
Name: target, dtype: int64

```

Out[227]: ((48938, 365), (48938,))

```

Ввод [228]: # Перерасчет дисбаланса классов
classes = np.unique(y_train_sampling)
weights = compute_class_weight(class_weight='balanced', classes=classes, y=y_train_sampling)
sampling_class_weights = dict(zip(classes, weights))
print(sampling_class_weights)

# Модель определения банкрота
bankrupt_model = CatBoostClassifier(eval_metric = "AUC", early_stopping_rounds=50, class_weights=sampling_class_weights,)
bankrupt_model.fit(X_train_sampling, y_train_sampling, eval_set=(X_val_bankrupt_custom, y_val_bankrupt_custom), plot=True, verbose=False)

```



Out[228]: <catboost.core.CatBoostClassifier at 0x131b54c1d88>

```

Ввод [229]: # Делаем предсказание модели (на тестовых данных) через predict_proba
y_pred_proba_bankrupt = bankrupt_model.predict_proba(X_test_bankrupt_custom)
# Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test_bankrupt_custom, y_score=y_pred_proba_bankrupt[:, 1])
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")
# Формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > best_thresh).astype(int)
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_bankrupt_custom, y_pred=y_pred_bankrupt)
roc_auc

```

Оптимальный порог: 0.4568146611964385



Out[229]: 0.6283773239581708

```

Ввод [243]: # формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > 0.419).astype(int)

print("Распределение предсказанного таргета: {sum(y_pred_bankrupt)/len(y_pred_bankrupt)}")
print("Распределение исходного таргета: {sum(y_test_bankrupt)/len(y_test_bankrupt)}")

metric_clients_accept, metric_clients_bankrupt = calc_metrics(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
metric_clients_accept, metric_clients_bankrupt

Распределение предсказанного таргета: 0.6437352245862884
Распределение исходного таргета: 0.2768321513002364
Одобрено 35% клиентов
Среди одобренных 14% клиентов-банкротов

```

Out[243]: (0.3562647754137116, 0.1493032514930325)

Результат сэмплирования данных

Использованы сэмплеры:

- RandomUnderSampler
- TomekLinks
- RandomOverSampler

Произведено сэмплирование данных, с целью решение проблемы дисбаланса классов. Однако

Однако сэмплирование не дало прироста скора.

Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Распределение предсказаний	Одобрено клиентов	% банкротов
1	Базовая модель	Простая модель	0.6575	0.5213	0.04	95%	26%
2	Модель №1 + дисбаланс классов	Учитываем в модели дисбаланс классов	0.6530	0.6104	0.42	57%	20%
3	Модель №2 + оптимальный threshold	Определяем для модели оптимальный threshold	0.6530	0.6187	0.56	43%	16%
4	Модель №3 + новые признаки	Сгенерированы новые признаки и выбран threshold (0.435). Без чистки признаков. Достигнут требуемый скор!	0.6616	0.6334	0.62	37,8%	14,4%
5	Модель №4 + чистка признаков	Убраны лишние признаки	0.6649	0.6260	0.61	38,4%	14,8%
6	Модель №5 + RandomUnderSampler	Уменьшаем мажорный класс до размера минорного	0.6624	0.6182	0.66	33,1%	14,8%
7	Модель №5 + TomekLinks	Убираем "выбросы" из мажорного класса	0.6644	0.6250	0.64	35,6%	14,9%
8	Модель №5 + RandomOverSampler	Увеличиваем минорный класс до размера мажорного	0.6615	0.6283	0.64	35,6%	14,9%

Лучшие результаты у Модели №5

Подбор гиперпараметров

```

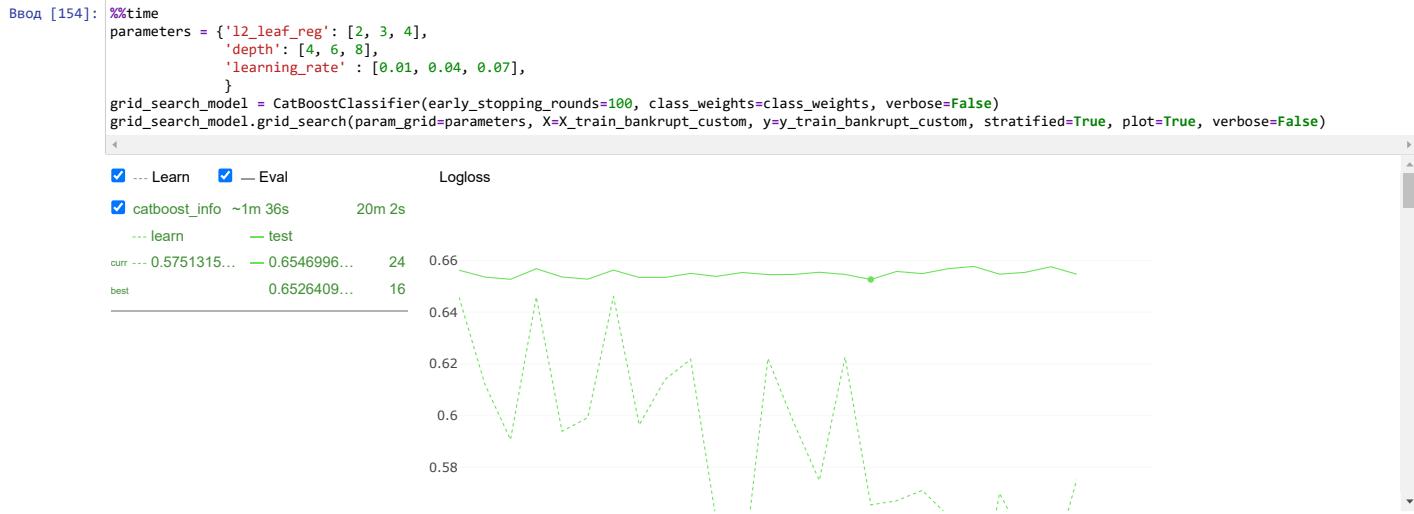
Ввод [153]: # Смотрим текущие значения параметров модели
bankrupt_model.get_all_params()

```

```

Out[153]: {'nan_mode': 'Min',
 'eval_metric': 'AUC',
 'iterations': 1000,
 'sampling_frequency': 'PerTree',
 'leaf_estimation_method': 'Newton',
 'od_pval': 0,
 'grow_policy': 'SymmetricTree',
 'penalties_coefficient': 1,
 'boosting_type': 'Plain',
 'model_shrink_mode': 'Constant',
 'feature_border_type': 'GreedyLogSum',
 'bayesian_matrix_reg': 0.1000000149011612,
 'eval_fraction': 0,
 'force_unit_auto_pair_weights': False,
 'l2_leaf_reg': 3,
 'random_strength': 1,
 'od_type': 'Iter',
 'rsm': 1,
 'boost_from_average': False,
 '...': ...
}

```

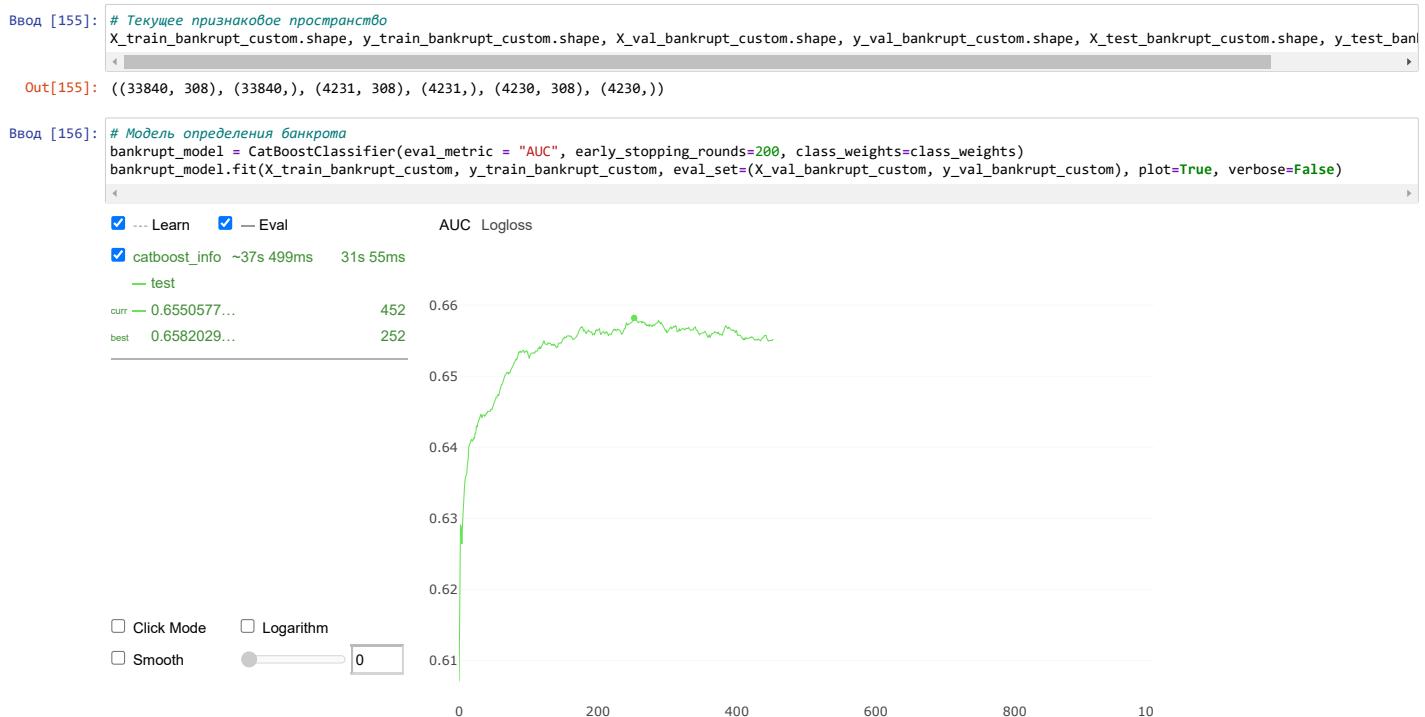


Выход анализа подбора гиперпараметров:

- Значения по умолчанию и параметры которые подбираются автоматически моделью, дают оптимальный скор
- Дополнительно был проведен анализ большей части настраиваемых параметров, но значимого роста скора не наблюдалось

В связи с этим использование механизмов подбора параметров типа optuna также не имеет смысла. Целесообразно делать акцент на данных

Анализ признакового пространства



Out[156]: <catboost.core.CatBoostClassifier at 0x1319a32c188>

```

Ввод [157]: # Делаем предсказание модели (на тестовых данных) через predict_proba
y_pred_proba_bankrupt = bankrupt_model.predict_proba(X_test_bankrupt_custom)
# Определяем оптимальный порог как поиск максимального значения для разницы tpr - fpr
fpr, tpr, thresholds = roc_curve(y_true=y_test_bankrupt_custom, y_score=y_pred_proba_bankrupt[:, 1])
best_thresh = thresholds[np.argmax(tpr - fpr)]
print(f"Оптимальный порог: {best_thresh}")
# Формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > best_thresh).astype(int)
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_bankrupt_custom, y_pred=y_pred_bankrupt)
roc_auc

```

Оптимальный порог: 0.4592396689447541



Out[157]: 0.6197541434620971

```

Ввод [158]: # формируем итоговое предсказание с учетом выбранного порога
y_pred_bankrupt = (y_pred_proba_bankrupt[:, 1] > 0.43).astype(int)

print("Распределение предсказанного таргета: {sum(y_pred_bankrupt)/len(y_pred_bankrupt)}")
print("Распределение исходного таргета: {sum(y_test_bankrupt)/len(y_test_bankrupt)}")

metric_clients_accept, metric_clients_bankrupt = calc_metrics(y_true=y_test_bankrupt, y_pred=y_pred_bankrupt)
metric_clients_accept, metric_clients_bankrupt

Распределение предсказанного таргета: 0.6153664302600473
Распределение исходного таргета: 0.2768321513002364
Одобрено 38% клиентов
Среди одобренных 16% клиентов-банкротов

```

Out[158]: (0.3846335697399527, 0.16103257529194837)

Ввод []:

Ввод [159]: shap.initjs()

```

Ввод [160]: explainer = shap.TreeExplainer(bankrupt_model)
shap_values=explainer.shap_values(Pool(X_train_bankrupt_custom, y_train_bankrupt_custom))

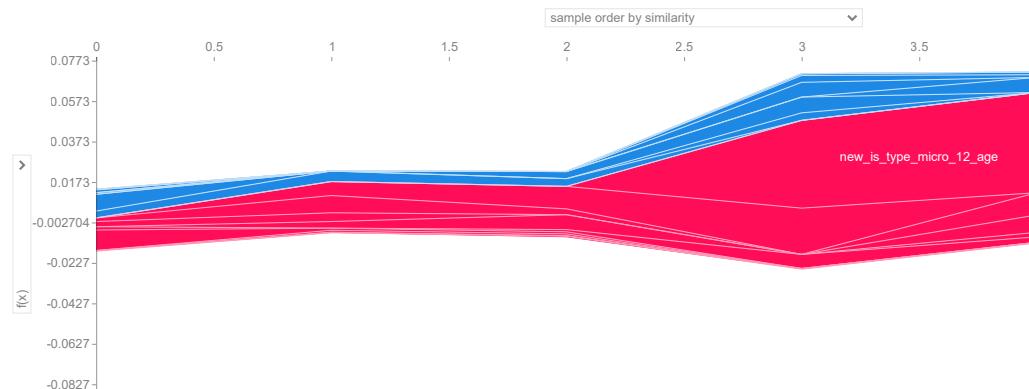
```

```

Ввод [161]: shap.force_plot(explainer.expected_value, shap_values[:5,:10], X_train_bankrupt_custom.iloc[:5,:10])

```

Out[161]:



```

Ввод [162]: shap.force_plot(explainer.expected_value, shap_values[3, :], X_train_bankrupt_custom.iloc[3, :])

```

Out[162]:



Ввод [163]: shap_values

```

Out[163]: array([[-4.24130350e-05, -1.31626105e-03, 2.66069552e-03, ...,
   -1.48010843e-02, 2.24166262e-03, 2.97415736e-02],
  [-2.99133002e-04, 1.01218119e-02, 1.63341573e-03, ...,
   -3.66022990e-03, -2.59819855e-03, -1.09319245e-03],
  [-4.24130350e-05, -8.34791951e-04, 6.96876962e-03, ...,
   -3.99890498e-03, 2.93766492e-03, -3.46988278e-03],
  ...,
  [ 5.17277412e-04, -8.34791951e-04, 1.05563818e-03, ...,
   2.38305226e-02, 2.66237854e-03, -1.86212279e-03],
  [ 4.57887768e-04, 1.03419783e-03, 1.33988871e-03, ...,
   -3.05103286e-03, -1.76794942e-04, -5.71081861e-04],
  [ 4.57887768e-04, 1.03419783e-03, 6.19781964e-03, ...,
   -1.37953525e-03, -3.14571323e-04, -2.33123152e-03]])

```

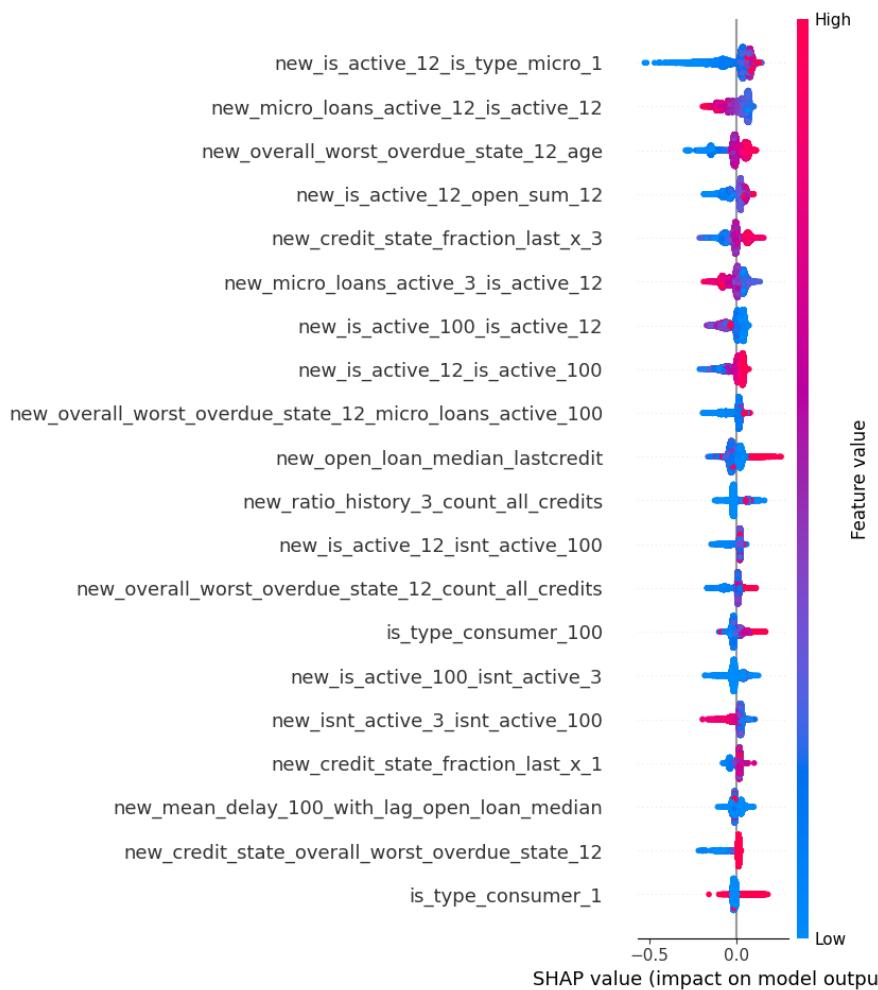
```
Ввод [164]: result_shape = pd.DataFrame(shap_values, columns=X_train_bankrupt_custom.columns)
vals = np.abs(result_shape.values).mean(0)
shap_importance = pd.DataFrame(list(zip(X_train_bankrupt_custom.columns, vals)), columns=['col_name', 'feature_importance_vals'])
shap_importance.sort_values(by=['feature_importance_vals'], ascending=False, inplace=True)
shap_importance.head(20)
```

Out[164]:

	col_name	feature_importance_vals
260	new_is_active_12_is_type_micro_1	0.076926
220	new_micro_loans_active_12_is_active_12	0.053800
265	new_overall_worst_overdue_state_12_age	0.043158
278	new_is_active_12_open_sum_12	0.041442
223	new_credit_state_fraction_last_x_3	0.040492
155	new_micro_loans_active_3_is_active_12	0.037452
67	new_is_active_100_is_active_12	0.034342
150	new_is_active_12_is_active_100	0.031733
184	new_overall_worst_overdue_state_12_micro_loans...	0.028510
266	new_open_loan_median_lastcredit	0.028006
225	new_ratio_history_3_count_all_credits	0.026521

```
Ввод [165]: shap.summary_plot(shap_values, X_train_bankrupt_custom, max_display=20, auto_size_plot=True)
```

auto_size_plot=False is deprecated and is now ignored! Use plot_size=None instead.



Анализ признаков

Как читать график shap:

- Значения слева от центральной вертикальной линии — это negative класс (0), справа — positive (1)
- Каждая точка, это одно наблюдение, чем толще линия на графике, тем больше таких точек наблюдения
- Чем краснее точки на графике, тем выше значения фичи в ней (например возраст 10 лет будет синим, а 70 лет красным)

Согласно полученному графику, наиболее важные признаки:

№	Название	Описание	Интерпретация
1	new_is_active_12_is_type_micro_1	Отношение: "Количество всех активных кредитов, открытых за последние 12 месяцев" к "Количество кредитов типа "микрокредит", открытых за последний месяц"	Если активные кредиты в основном имеют тип "микрокредит", то меньше вероятность, что будет банкротом
2	new_is_active_12_is_active_100	Отношение: "Количество всех активных кредитов, открытых за последние 12 месяцев" к "Количество активных кредитов, открытых за все время"	Если большая часть действующих кредитов набрана в последние 12 месяцев, то высока вероятность банкротства
3	new_is_active_12_open_sum_12	Отношение: "Количество всех активных кредитов, открытых за последние 12 месяцев" к "Активная сумма кредитов, взятых за последние 12 месяцев"	Один большой кредит лучше, чем 10 маленьких. Если человек просто взял большой кредит, у него меньше шансов быть банкротом, нежели человек который набирает ту же сумму за счет открытия новых кредитов
4	new_micro_loans_active_12_is_active_12	Отношение: "Активная сумма микрокредитов, открытых за последние 12 месяцев" к "Количество всех активных кредитов, открытых за последние 12 месяцев"	Чем выше значение, тем меньше вероятность банкротства. Если активные кредиты (по сумме) в основном имеют тип "микрокредит", то меньше вероятность, что будет банкротом

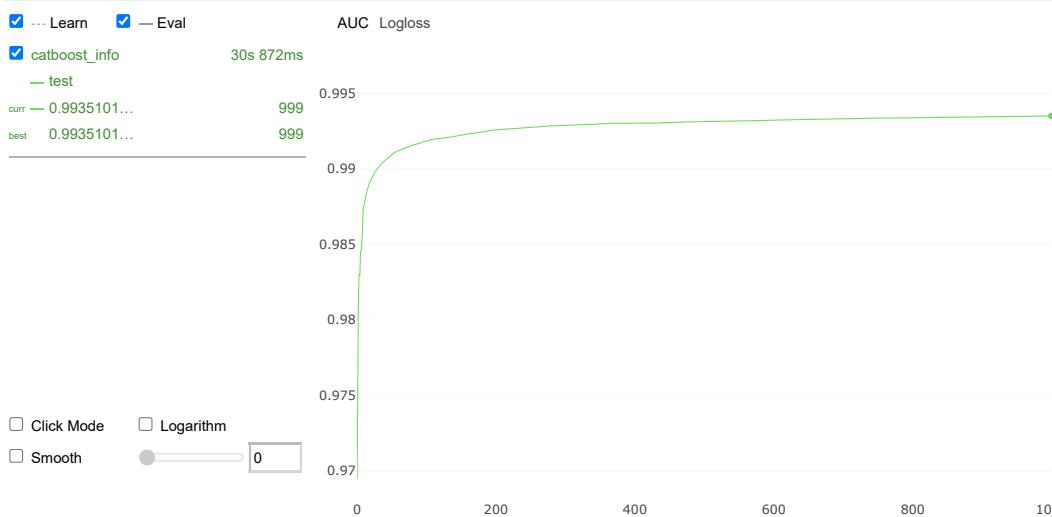
№	Название	Описание	Интерпретация
5	new_credit_state_fraction_last_x_1	Произведение: "Доля кредитов, взятых за последний месяц, относительно всех кредитов истории" на "Статус по просроченным платежам за 12 месяцев (overall_worst_overdue_state_12)"	Чем меньше значение, тем меньше вероятность банкротства. По сути это взвешивание одного показателя за счет другого
6	new_credit_state_fraction_last_x_3	Произведение: "Доля кредитов, взятых за последние 3 месяца, относительно всех кредитов истории" на "Статус по просроченным платежам за 12 месяцев (overall_worst_overdue_state_12)"	Чем меньше значение, тем меньше вероятность банкротства. По сути это взвешивание одного показателя за счет другого
7	new_micro_loans_active_3_is_active_12	Отношение: "Активная сумма микрокредитов, открытых за последние 3 месяца" к "Количество всех активных кредитов, открытых за последние 12 месяцев"	Чем выше значение, тем меньше вероятность банкротства. Если активные кредиты (по сумме) в основном имеют тип "микрокредит", то меньше вероятность, что будет банкротом
8	new_micro_loans_active_100_is_active_100	Отношение: "Активная сумма микрокредитов, открытых за всё время" к "Количество активных кредитов, открытых за все время"	Чем выше значение, тем меньше вероятность банкротства. Если активные кредиты (по сумме) в основном имеют тип "микрокредит", то меньше вероятность, что будет банкротом
9	new_is_active_12_is_type_micro_3	Отношение: "Количества всех активных кредитов, открытых за последние 12 месяцев" к "Количество кредитов типа "микрокредит", открытых за последние 3 месяца"	Если активные кредиты в основном имеют тип "микрокредит", то меньше вероятность, что будет банкротом
10	new_isnt_active_100_overall_worst_overdue_state_12	Отношение: "Доля не возвращенных кредитов относительно всех кредитов, взятых за все время" к "Статус по просроченным платежам за 12 месяцев (overall_worst_overdue_state_12)"	Чем меньше значение, тем меньше вероятность банкротства.
11	new_overall_worst_overdue_state_12_count_all_credits	Отношение: "Статус по просроченным платежам за 12 месяцев (overall_worst_overdue_state_12)" к "Количество всех кредитов в истории"	Чем меньше значение, тем меньше вероятность банкротства.
12	new_open_loan_median_lastcredit	Отношение: "Медиана, взята по количеству дней между открытием предыдущего и следующего микрокредита" к "Время в днях, которое прошло с момента открытия последнего кредитного продукта"	Чем выше значение, тем выше вероятность банкротства. Требуется более детальный анализ. Вероятно есть зависимость, такого рода, что чем ближе к завершению кредита, тем меньше вероятность банкротства.
13	new_is_active_12_count_all_credits	Отношение: "Количества всех активных кредитов, открытых за последние 12 месяцев" к "Количество всех кредитов в истории"	Чем ниже, тем меньше вероятность банкротства. Т.е. если у человека всего было 100 кредитов и сейчас открыто 2, значит велика вероятность, что он их умешно погасит
14	is_type_consumer_100	Базовый признак: "Количество кредитов типа "потребительский кредит", открытых за всё время"	Чем чаще человек пользуется кредитами, тем выше вероятность банкротства
15	age	Базовый признак: "Возраст"	Чем старше люди, тем ниже вероятность банкротства и наоборот, чем младше тем выше.

Выход:

- Практически все сгенерированные признаки однозначно интерпретируются под целевой таргет

Дополнительно проанализируем важные фичи из базовых для модели определения выдавать кредит или нет

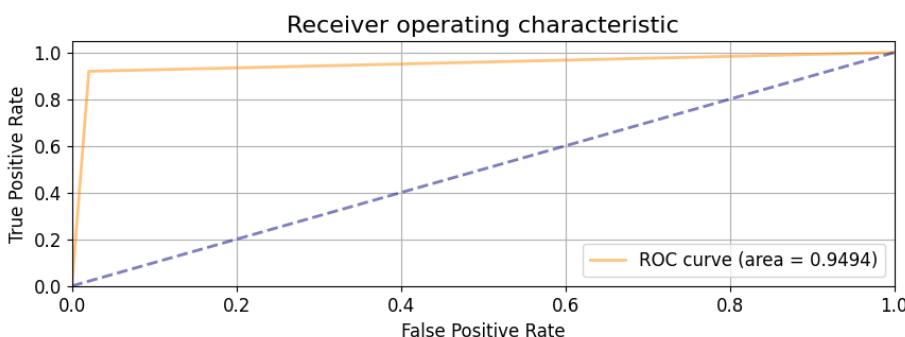
```
Ввод [166]: # Обучаем модель принятия решения выдавать кредит или нет
accept_model.fit(X_train_accept, y_train_accept, eval_set=(X_val_accept, y_val_accept), plot=True, verbose=False)
```



```
Out[166]: <catboost.core.CatBoostClassifier at 0x13199c01dc8>
```

```
Ввод [167]: # Для расчета ROC-AUC на baseline модели используем тестовые данные
y_pred_accept = accept_model.predict(X_test_accept)
```

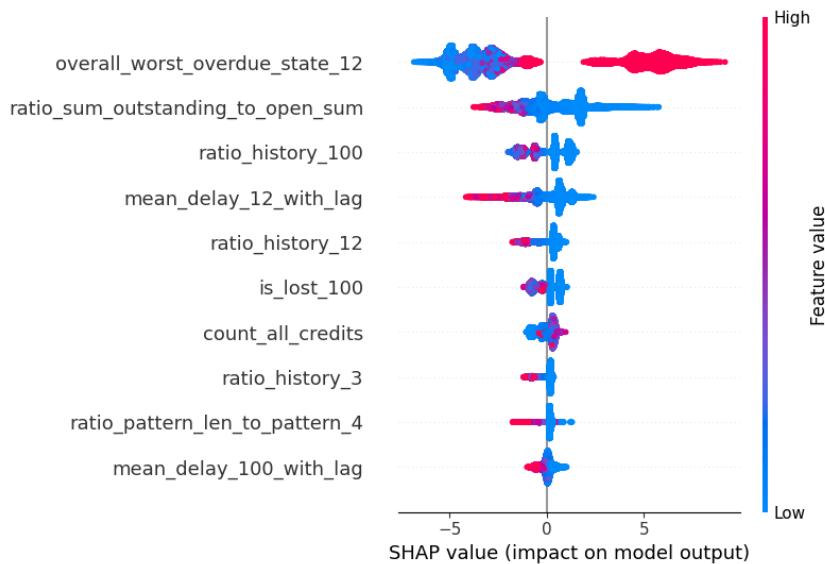
```
# Строим график ROC-AUC
roc_auc = plot_roc_auc(y_true=y_test_accept, y_pred=y_pred_accept)
roc_auc
```



```
Out[167]: 0.9493824178949882
```

```
Ввод [168]: # Важность признаков
explainer_accept = shap.TreeExplainer(accept_model)
shap_values_accept=explainer_accept.shap_values(Pool(X_train_accept, y_train_accept))
```

```
Ввод [169]: shap.summary_plot(shap_values_accept, X_train_accept, max_display=10)
```



Выход:

- Ключевое решение о выдаче займа решает признак: "overall_worst_overdue_state_12" (Статус по просроченным платежам за 12 месяцев).

Отчет

Выполнено:

1. Анализ данных

- Проведен анализ данных
- Обработаны пропуски и неверные значения
- Анализ значимости признаков. Построение матрицы корреляции
- Анализ значений близких к константе
- Профилирование признаков. Отчет профилирования в "ProfileReport_train.html"

2. Модели

- Построение двух базовых моделей: Прогнозирование выдачи займа (таргет: выдан займ). Прогнозирование банкротства (таргет: банкрот).
- Проведена балансировка классов
- Выбор порога (threshold) для достижения заданных метрик
- Генерации новых признаков (~1000 новых признаков)
- Исключение признаков не влияющих на таргет и сильно коррелирующих между собой
- Кластеризация данных, для визуального анализа
- Сэмплирование данных (в целях борьбы с дисбалансом)
- Подбор гиперпараметров
- Достигнуты на тестовых данных требования 35% клиентов при банкротстве среди одобренных не выше 15%

3. Анализ признаков Подробный анализ признакового пространства (с интерпретацией)

Рекомендации к данным:

- из описания "isnt_active_100 - Доля не возвращенных кредитов относительно всех кредитов, взятых за все время" Однако по данным максимальное значение 6743
- Параметр "ratio_sum_outstanding_to_open_sum" имеет значения +inf/-inf По описанию этот параметр представляет собой: "Отношение суммы просрочки по всем кредитам к сумме взятых кредитов за всю историю" Соответственно неопределенность может возникать, когда значения знаменателя стремятся к 0. Поэтому рекомендуются при расчете такого рода параметров предусматривать такие ситуации. Самый простой способ уйти от такого рода проблем заменить формулу: "x/y" на "x/(y+1)"
- Параметры:
 - overdue_loans_12
 - overdue_loans_3
 - ratio_overdue_loans_3_to_12 имеют константные значения, это говорит либо о некорректной выборке либо о том что эти условия для формирования этих признаков встречаются довольно редко, и они не дают значения в плюне информационного выигрыша
- Добавить id клиента, чтобы при агрегации различных списков понимать, что речь про одного человека. Это также позволит вводить дополнительный контроль поступаемых данных. Если в предоставленных данных указаны не сами клиенты, а только факты выдачи или отказа займа, тогда наличие ИД клиента обязательно! ИД позволит сделать формирование профиля клиента, и это можно использовать для более правильного обучения модели. Сейчас же приходится принимать допущения, что это разные люди.
- Описание для параметров micro_loans_active_* не дает представление о сущности этого признака, что значит активная, это сколько человек сейчас должен по всем микрозаймам, или это показатель как человек активно гасит кредит. Судя по корреляции чем больше это значение тем меньше вероятность банкротства
- Много признаков которые сильно склонены к одному числу как правило к 0. Например 97% значений это нули, такие признаки чаще всего не дают информационный выигрыш (если только у оставшихся 3% данных нет сильной корреляции с таргетом). При фининжениринге надо стараться привести к более представительным признакам, чтобы с их помощью можно было разделять пространство данных.
- Анализируя важность признаков, наблюдаем, что признак "overall_worst_overdue_state_12" является решающим для прогнозирования выдачи займа (при анализе корреляции это признак также был самым значимым). Для признака "overall_worst_overdue_state_12" нет описания, но судя по названию, это общий наихудший статус по просроченным платежам за 12 месяцев. Можно сделать следующие предположения по этому признаку:
 - Признак "overall_worst_overdue_state_12" содержит дополнительную информацию, которая не участвовала в остальных признаках. Т.е. этот признак содержит такую информацию которая не использовалась в других агрегационных признаках (например это статус благонадежности человека из внешней базы).
 - Это действительно очень удачно созданный признак, использующий известные данные. Например этот признак мог быть сгенерирован другой моделью машинного обучения.
 - Этот признак является решающим для принятия решения о выдаче займа. По условию: отказ в займе может быть по причине отказа финансовой организации или нежеланию самого клиента. Если допустить, что людей которые подали заявку на займ, а потом передумали меньшество, тогда большая доля отказов по причине отказа финансовой организации. Для отказов действует набор правил по которым происходит отказ клиентам, и в таком случае "overall_worst_overdue_state_12" является решающим правилом (наиажнейшим).
 - При формировании признака мог быть допущен лик данных (утечка таргета). Тогда он очень хорошо разделяет известные данные, но на неизвестных результат уже не будет таким же. Требуется описать данный параметр, и проанализировать его природу и понять, почему такое сильное влияние на прогноз выдачи займа.
- Данные представлены в виде агрегации данных, но в качестве развития необходимо понимать какие есть исходные данные, чтобы предложить какие дополнительные агрегации можно сделать
- Необходимо добавить дополнительное описание к данным, где отразить природу и особенности данных, в том числе:

- отразить могут ли в двух списках быть один и тот же человек или это всегда разные люди (если могут то с какой частотой).

10. Добавить фичу: время между закрытием предыдущего микрозайма и открытием нового

Результаты экспериментов:

№	Название	Описание	ROC-AUC на val	ROC-AUC на test	Распределение предсказаний	Одобрено клиентов	% банкротов
1	Базовая модель	Простая модель	0.6575	0.5213	0.04	95%	26%
2	Модель №1 + дисбаланс классов	Учитываем в модели дисбаланс классов	0.6530	0.6104	0.42	57%	20%
3	Модель №2 + оптимальный threshold	Определяем для модели оптимальный threshold	0.6530	0.6187	0.56	43%	16%
4	Модель №3 + новые признаки	Сгенерированы новые признаки и выбран threshold (0.435). Без чистки признаков. Достигнут требуемый скор!	0.6616	0.6334	0.62	37,8%	14,4%
5	Модель №4 + чистка признаков	Убраны лишние признаки	0.6649	0.6260	0.61	38,4%	14,8%
6	Модель №5 + RandomUnderSampler	Уменьшаем мажорный класс до размера минорного	0.6624	0.6182	0.66	33,1%	14,8%
7	Модель №5 + TomekLinks	Убираем "выбросы" из мажорного класса	0.6644	0.6250	0.64	35,6%	14,9%
8	Модель №5 + RandomOverSampler	Увеличиваем минорный класс до размера мажорного	0.6615	0.6283	0.64	35,6%	14,9%

Лучшие результаты у Модели №5

Ввод []: