Выполнил: Соковнин Игорь

Спортивный анализ данных. Платформа Kaggle

Урок 5. Feature Engineering, Feature Selection, part I

Домашнее задание:

Продолжим работу с данными, которые были использованы в ДЗ2 и 3, продолжим решать задачу обнаружения мошеннических транзакций, что позволит получить полное решение задачи / полный пайплайн.

Задание 0: Выбрать любую модель машинного обучения и зафиксировать любой тип валидации. Обучить базовую модель и зафиксировать базовое качество модели. В каждом следующем задании нужно будет обучить выбранную модель и оценивать ее качество на зафиксированной схеме валидации. После каждого задания, требуется сделать вывод о достигаемом качестве модели, по сравнению с качестом из предыдущего шага.

Задание 1: Признак TransactionDT - это смещение в секундах относительно базовой даты. Базовая дата - 2017-12-01, преобразовать признак TransactionDT в datetime, прибавив к базовой дате исходное значение признака. Из полученного признака выделить год, месяц, день недели, час, день.

Задание 2: Сделать конкатенацию признаков

- card1 + card2;
- card1 + card2 + card_3 + card_5;
- card1 + card2 + card_3 + card_5 + addr1 + addr2

Рассматривать их как категориальных признаки.

Задание 3: Сделать *FrequencyEncoder* для признаков *card1* - *card6, addr1, addr2*.

Задание 4: Создать признаки на основе отношения: TransactionAmt к вычисленной статистике. Статистика - среднее значение / стандартное отклонение TransactionAmt, сгруппированное по card1 - card6, addr1, addr2, и по признакам, созданным в задании 2.

Задание 5: Создать признаки на основе отношения: D15 к вычисленной статистике. Статистика - среднее значение / стандартное отклонение D15, сгруппированное по card1 - card6, addr1, addr2, и по признакам, созданным в задании 2.

Задание 6: Выделить дробную часть и целую часть признака TransactionAmt в два отдельных признака. После создать отдельных признак - логарифм от TransactionAmt

Задание 7 (опция): Выполнить предварительную подготовку / очистку признаков P_emaildomain и R_emaildomain (что и как делать - остается на ваше усмотрение) и сделать Frequency Encoding для очищенных признаков.

Вывод по заданию:

Улучшение модели по сравнению с бозовым решением дали созданные признаки из **Задания 2**, **Задания 3** и **Задания 6**. Требуется дальнейший анализ.

Задание 0 (без обработки):

- bestTest = 0.8827161236
- bestIteration = 419

Задание 1:

- bestTest = 0.8812417137
- bestIteration = 455

Вывод:

- Добавление новых признаков (Задание 1) не дало улучшения качества модели по сравнению с базовым решением.

Задание 2:

- bestTest = 0.9216976237
- bestIteration = 557

Вывод:

- Добавление новых признаков (Задание 2) значительно улучшило качество модели по сравнению с базовым решением.

Задание 3:

- bestTest = 0.9180509792
- bestIteration = 506

Вывод:

- Добавление новых признаков (Задание 3) значительно улучшило качество модели по сравнению с базовым решением.

Задание 4:

- bestTest = 0.8728269204
- bestIteration = 382

Вывод:

- Добавление новых признаков (Задание 4) не дало улучшения качества модели по сравнению с базовым решением.

Задание 5:

- bestTest = 0.8709918449
- bestIteration = 483

Вывод:

- Добавление новых признаков (Задание 5) не дало улучшения качества модели по сравнению с базовым решением.

Задание 6:

- bestTest = 0.8828945346
- bestIteration = 443

Вывод:

- Добавление новых признаков (Задание 6) незначительно качества модели по сравнению с базовым решением.

Подключение библиотек и скриптов

В [1]:

```
import datetime
import warnings
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

# Модель
import xgboost as xgb
import catboost as cb

# Метрика
from sklearn.metrics import roc_auc_score, auc
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split, cross_val_sco

warnings.simplefilter("ignore")
%matplotlib inline
```

В [2]:

```
# разварачиваем выходной дисплей, чтобы увидеть больше столбцов и строк а pandas DataFrame
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

В [3]:

```
def reduce_mem_usage(df):
    '''Сокращение размера датафрейма за счёт изменения типа данных'''

    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            df[col] = df[col].astype('category')

    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

    return df
```

В [4]:

```
!dir
```

```
 Том в устройстве C имеет метку Новый том
 Серийный номер тома: 6E3D-C99D

 Содержимое папки C:\Users\sil\Desktop\Python_for_DataSience\Спортивный анал
из данных. Платформа Kaggle II\Урок 5. Feature Engineering, Feature Selectio
n, part I\HW

29.03.2021  11:48    <DIR>          .
29.03.2021  11:48    <DIR>          ..
27.03.2021  14:05    <DIR>          .ipynb_checkpoints
28.03.2021  14:08    <DIR>          catboost_info
29.03.2021  02:02           239 022 lesson_5_hw - 2021-03-29.ipynb
28.03.2021  16:49           163 768 lesson_5_hw 2021-03-28 CatBoost.ipynb
28.03.2021  13:39           118 506 lesson_5_hw 2021-03-28 XGBoost.ipynb
29.03.2021  11:48           182 287 lesson_5_hw.ipynb
               4 файлов        703 583 байт
               4 папок  70 710 841 344 байт свободно
```

В [5]:

```
# input
TRAIN_DATASET_PATH = '../../data/assignment_2_train.csv'
TEST_DATASET_PATH = '../../data/assignment_2_test.csv'
```

Загрузка данных

В [6]:

```
# Тренировочные данные
# train = pd.read_csv(TRAIN_DATASET_PATH, header = none)  # если надо скрыть названия столб
train = pd.read_csv(TRAIN_DATASET_PATH)
df_train =reduce_mem_usage(train)  # Уменьшаем размер данныхМ
df_train.head(2)
```

```
Memory usage of dataframe is 541.08 MB
Memory usage after optimization is: 262.48 MB
Decreased by 51.5%
```

Out[6]:

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	
0	2987000	0	86400	68.5	W	13926	NaN	150.0	
1	2987001	0	86401	29.0	W	2755	404.0	150.0	ma

В [7]:

```
# Тестовые данные
leaderboard = pd.read_csv(TEST_DATASET_PATH)
df_test =reduce_mem_usage(leaderboard)  # Уменьшаем размер данных

df_test.head(2)
```

```
Memory usage of dataframe is 300.60 MB
Memory usage after optimization is: 145.83 MB
Decreased by 51.5%
```

Out[7]:

	TransactionID	isFraud	TransactionDT	TransactionAmt	ProductCD	card1	card2	card3	ca
0	3287000	1	7415038	226.0	W	12473	555.0	150.0	\
1	3287001	0	7415054	3072.0	W	15651	417.0	150.0	\

Числовых признаки

В [8]:

```
# Общее количество записей в датафрейме = 180 000
# Исключаем такие поля содержащие меньше 100 000 значений,
# из предполажения, что значение этих полей несущественно (всегда можно этот параметр прова

numerical_features = [
'TransactionID',  # Индекс
'isFraud',  # Целевой параметр
'TransactionDT',  # Временя совершения транзакции
'TransactionAmt',  # Сумма транзакции
'card1',
'card2',
'card3',
'card5',
'addr1',
'addr2',
'C1',
'C2',
'C3',
'C4',
'C5',
'C6',
'C7',
'C8',
'C9',
'C10',
'C11',
'C12',
'C13',
'C14',
'D1',
'D4',
'D10',
#'D11', ## <  50 000
'D15',
'V12',
'V13',
'V14',
'V15',
'V16',
'V17',
'V18',
'V19',
'V20',
'V21',
'V22',
'V23',
'V24',
'V25',
'V26',
'V27',
'V28',
'V29',
'V30',
'V31',
'V32',
'V33',
'V34',
'V35',
'V36',
```

```
'V37',
'V38',
'V39',
'V40',
'V41',
'V42',
'V43',
'V44',
'V45',
'V46',
'V47',
'V48',
'V49',
'V50',
'V51',
'V52',
'V53',
'V54',
'V55',
'V56',
'V57',
'V58',
'V59',
'V60',
'V61',
'V62',
'V63',
'V64',
'V65',
'V66',
'V67',
'V68',
'V69',
'V70',
'V71',
'V72',
'V73',
'V74',
'V75',
'V76',
'V77',
'V78',
'V79',
'V80',
'V81',
'V82',
'V83',
'V84',
'V85',
'V86',
'V87',
'V88',
'V89',
'V90',
'V91',
'V92',
'V93',
'V94',
'V95',
'V96',
'V97',
```

```
'V98',
'V99',
'V100',
'V101',
'V102',
'V103',
'V104',
'V105',
'V106',
'V107',
'V108',
'V109',
'V110',
'V111',
'V112',
'V113',
'V114',
'V115',
'V116',
'V117',
'V118',
'V119',
'V120',
'V121',
'V122',
'V123',
'V124',
'V125',
'V126',
'V127',
'V128',
'V129',
'V130',
'V131',
'V132',
'V133',
'V134',
'V135',
'V136',
'V137',
'V280',
'V281',
'V282',
'V283',
'V284',
'V285',
'V286',
'V287',
'V288',
'V289',
'V290',
'V291',
'V292',
'V293',
'V294',
'V295',
'V296',
'V297',
'V298',
'V299',
'V300',
```

```
'V301',
'V302',
'V303',
'V304',
'V305',
'V306',
'V307',
'V308',
'V309',
'V310',
'V311',
'V312',
'V313',
'V314',
'V315',
'V316',
'V317',
'V318',
'V319',
'V320',
'V321'
]
```

Обрабатка категориальные признаков

В [9]:

```
catigorical_features = [
'ProductCD',  # 180000 non-null  category
'card4',  # 179992 non-null  category
'card6',  # 179993 non-null  category
'P_emaildomain',  # 151560 non-null  category
'R_emaildomain',  # 60300 non-null   category
'M1',  # 61749 non-null   category
'M2',  # 61749 non-null   category
'M3',  # 61749 non-null   category
'M4',  # 83276 non-null   category
'M5',  # 61703 non-null   category
'M6',  # 105652 non-null  category
'M7',  # 31652 non-null   category
'M8',  # 31652 non-null   category
'M9'  # 31652 non-null   category
]
```

Подготовка тренировочных данных

В [10]:

```
data = []
data = df_train[numerical_features + catigorical_features]

# заполняем пропуски в категориалиных признаках
for feature in catigorical_features:
    data[feature] = data[feature].cat.add_categories('Unknown')
    data[feature].fillna('Unknown', inplace =True)

# Каждой категории сопоставляет целое число (номер категории) - https://dyakonov.org/2016/0
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for cat_colname in data[catigorical_features].columns:
    le.fit(data[cat_colname])
    data[cat_colname+'_le'] = le.transform(data[cat_colname])

target = df_train["isFraud"]
```

В [11]:

```
df_train_new = data
#df_train_new = df_train_new.drop(catigorical_features, axis=1)
# df_train_new.columns
```

В [12]:

```
# df_train_new = df_train_new.drop(["TransactionID", "TransactionDT", "isFraud"], axis=1)
```

В [13]:

```
catigorical_features_new = ['ProductCD_le', 'card4_le', 'card6_le', 'R_emaildomain_le',
                      'M1_le', 'M2_le', 'M3_le', 'M4_le', 'M5_le', 'M6_le', 'M7_le', 'M8_le
```

Подготовка тестовых данных

В [14]:

```
data = []
data = df_test[numerical_features + catigorical_features]

# заполняем пропуски в категориалиных признаках
for feature in catigorical_features:
    data[feature] = data[feature].cat.add_categories('Unknown')
    data[feature].fillna('Unknown', inplace =True)

# Каждой категории сопоставляет целое число (номер категории) - https://dyakonov.org/2016/0
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
for cat_colname in data[catigorical_features].columns:
    le.fit(data[cat_colname])
    data[cat_colname+'_le'] = le.transform(data[cat_colname])

#target = df_train["isFraud"]

df_test_new = data
#f_test_new = df_test_new.drop(catigorical_features, axis=1)
df_test_new = df_test_new.drop(["TransactionID"], axis=1)
```

Задание 0:

Выбрать любую модель машинного обучения и зафиксировать любой тип валидации. Обучить базовую модель и зафиксировать базовое качество модели. В каждом следующем задании нужно будет обучить выбранную модель и оценивать ее качество на зафиксированной схеме валидации. После каждого задания, требуется сделать вывод о достигаемом качестве модели, по сравнению с качестом из предыдущего шага.

Hold-Out разбиение (Hold-Out валидация)

В [15]:

```
data = df_train_new
target = data["isFraud"]
#data = data.drop(["TransactionID", "TransactionDT", "isFraud"], axis=1)
data = data.drop(["TransactionID", "isFraud"], axis=1)
```

В [16]:

```
x_train, x_test = train_test_split(
    data, train_size=0.75, random_state=27
)
y_train, y_test = train_test_split(
    target, train_size=0.75, random_state=27
)
print("x_train.shape = {} rows, {} cols".format(*x_train.shape))
print("x_test.shape = {} rows, {} cols".format(*x_test.shape))
```

```
x_train.shape = 135000 rows, 222 cols
x_test.shape = 45000 rows, 222 cols
```

В [17]:

```
model = {}
train_scores = pd.DataFrame({"target": y_train})
test_scores = pd.DataFrame({"target": y_test})
```

В [18]:

```
#x_train.head(2)
#print(x_train.info())
#x_test.head(2)
#print(x_train.info())
```

XGBoost на числовых признаках

В [19]:

```
xgb_numerical_features = numerical_features.copy()  # Создаём копию списка
xgb_numerical_features.remove('isFraud')
xgb_numerical_features.remove('TransactionID')
#xgb_numerical_features.remove('TransactionDT')
```

В [20]:

```
xgb_params = {
    "booster": "gbtree",
    "objective": "binary:logistic",
    "eval_metric": "auc",
    "n_estimators": 1000,
    "learning_rate": 0.1,
    "reg_lambda": 10,
    "max_depth": 4,
    "gamma": 10,
    "nthread": 6,
    "seed": 27
}

eval_sets= [
    (x_train[xgb_numerical_features], y_train),
    (x_test[xgb_numerical_features], y_test)
]
```

В [21]:

```
xgb_model = xgb.XGBClassifier(**xgb_params)

xgb_model.fit(
    y=y_train,
    X=x_train[xgb_numerical_features],
    early_stopping_rounds=50,
    eval_set=eval_sets,
    eval_metric="auc",
    verbose=10
)

model["XGBoost_gbtree_num_features"] = xgb_model
```

```
[0]	validation_0-auc:0.75709	validation_1-auc:0.74768
[10]	validation_0-auc:0.80798	validation_1-auc:0.79743
[20]	validation_0-auc:0.84054	validation_1-auc:0.82946
[30]	validation_0-auc:0.87095	validation_1-auc:0.86259
[40]	validation_0-auc:0.88017	validation_1-auc:0.87050
[50]	validation_0-auc:0.88913	validation_1-auc:0.87711
[60]	validation_0-auc:0.89620	validation_1-auc:0.88277
[70]	validation_0-auc:0.90007	validation_1-auc:0.88530
[80]	validation_0-auc:0.90428	validation_1-auc:0.88827
[90]	validation_0-auc:0.90599	validation_1-auc:0.88941
[100]	validation_0-auc:0.90792	validation_1-auc:0.89099
[110]	validation_0-auc:0.91035	validation_1-auc:0.89305
[120]	validation_0-auc:0.91163	validation_1-auc:0.89392
[130]	validation_0-auc:0.91163	validation_1-auc:0.89392
[140]	validation_0-auc:0.91163	validation_1-auc:0.89392
[150]	validation_0-auc:0.91163	validation_1-auc:0.89392
[160]	validation_0-auc:0.91163	validation_1-auc:0.89392
[166]	validation_0-auc:0.91163	validation_1-auc:0.89392
```

В [22]:

```
train_scores["XGBoost_gbtree_num_features"] = xgb_model.predict_proba(x_train[xgb_numerical
test_scores["XGBoost_gbtree_num_features"] = xgb_model.predict_proba(x_test[xgb_numerical_f
```

В [23]:

```
train_scores
```

Out[23]:

	target	XGBoost_gbtree_num_features
141582	0	0.012777
131503	0	0.013938
173925	0	0.010473
177012	0	0.002903
69958	0	0.010226
...	...	...
4848	0	0.007321
14879	0	0.007348
36680	0	0.009374
118456	0	0.003609
5139	0	0.005172

135000 rows × 2 columns

CatBoost на числовых признаках

В [24]:

```
import catboost as cb
```

В [25]:

```
cb_params = {
    "n_estimators": 1000,
    "loss_function": "Logloss",
    "eval_metric": "AUC",
    "task_type": "CPU",
    #"max_bin": 20,
    "verbose": 10,
    "max_depth": 6,
    "l2_leaf_reg": 100,
    "early_stopping_rounds": 50,
    "thread_count": 6,
    "random_seed": 42
}

eval_sets= [
    (x_train[xgb_numerical_features], y_train),
    (x_test[xgb_numerical_features], y_test)
]
```

В [26]:

```
cb_model = cb.CatBoostClassifier(**cb_params)
cb_model.fit(x_train[xgb_numerical_features], y_train, eval_set=eval_sets)
```

```
0:	test: 0.6536584 test1: 0.6509021	best: 0.6509021 (0)	tota
l: 222ms	remaining: 3m 41s
10:	test: 0.7782015 test1: 0.7634376	best: 0.7687531 (7)	tota
l: 1.21s	remaining: 1m 48s
20:	test: 0.8199294 test1: 0.8049216	best: 0.8049216 (20)	tota
l: 2.33s	remaining: 1m 48s
30:	test: 0.8359524 test1: 0.8252769	best: 0.8252769 (30)	tota
l: 3.06s	remaining: 1m 35s
40:	test: 0.8482519 test1: 0.8384418	best: 0.8384418 (40)	tota
l: 3.71s	remaining: 1m 26s
50:	test: 0.8514080 test1: 0.8403066	best: 0.8403928 (47)	tota
l: 4.27s	remaining: 1m 19s
60:	test: 0.8539646 test1: 0.8411689	best: 0.8411689 (60)	tota
l: 4.82s	remaining: 1m 14s
70:	test: 0.8557345 test1: 0.8428050	best: 0.8431332 (69)	tota
l: 5.38s	remaining: 1m 10s
80:	test: 0.8603778 test1: 0.8481003	best: 0.8481003 (80)	tota
l: 5.96s	remaining: 1m 7s
90:	test: 0.8657723 test1: 0.8548091	best: 0.8548091 (90)	tota
l: 6.55s	remaining: 1m 5s
100:	test: 0.8678208 test1: 0.8568615	best: 0.8568615 (100)	tota
l: 7.08s	remaining: 1m 3s
110:	test: 0.8698583 test1: 0.8591075	best: 0.8591075 (110)	tota
l: 7.66s	remaining: 1m 1s
120:	test: 0.8716381 test1: 0.8608349	best: 0.8608349 (120)	tota
l: 8.24s	remaining: 59.8s
130:	test: 0.8728403 test1: 0.8624763	best: 0.8624942 (129)	tota
l: 8.82s	remaining: 58.5s
140:	test: 0.8741322 test1: 0.8640435	best: 0.8640435 (140)	tota
l: 9.43s	remaining: 57.4s
150:	test: 0.8754839 test1: 0.8646731	best: 0.8646731 (150)	tota
l: 10s	remaining: 56.5s
160:	test: 0.8769138 test1: 0.8658259	best: 0.8658259 (160)	tota
l: 10.6s	remaining: 55.3s
170:	test: 0.8789194 test1: 0.8682079	best: 0.8682079 (170)	tota
l: 11.2s	remaining: 54.3s
180:	test: 0.8802028 test1: 0.8697750	best: 0.8697750 (180)	tota
l: 11.8s	remaining: 53.3s
190:	test: 0.8821344 test1: 0.8718809	best: 0.8718809 (190)	tota
l: 12.4s	remaining: 52.5s
200:	test: 0.8834463 test1: 0.8735594	best: 0.8735594 (200)	tota
l: 13s	remaining: 51.5s
210:	test: 0.8846544 test1: 0.8744413	best: 0.8744413 (210)	tota
l: 13.5s	remaining: 50.6s
220:	test: 0.8852716 test1: 0.8753191	best: 0.8753225 (219)	tota
l: 14.1s	remaining: 49.7s
230:	test: 0.8860888 test1: 0.8760634	best: 0.8760634 (230)	tota
l: 14.7s	remaining: 48.8s
240:	test: 0.8867219 test1: 0.8765119	best: 0.8765119 (240)	tota
l: 15.2s	remaining: 47.9s
250:	test: 0.8871234 test1: 0.8769862	best: 0.8769862 (250)	tota
l: 15.8s	remaining: 47s
260:	test: 0.8875339 test1: 0.8774721	best: 0.8774721 (260)	tota
l: 16.3s	remaining: 46.2s
270:	test: 0.8881798 test1: 0.8780760	best: 0.8780760 (270)	tota
```

```
l: 16.9s        remaining: 45.4s
280:    test: 0.8886019 test1: 0.8783551        best: 0.8783551 (280)   tota
l: 17.4s        remaining: 44.6s
290:    test: 0.8890933 test1: 0.8787014        best: 0.8787014 (290)   tota
l: 18s  remaining: 43.8s
300:    test: 0.8895511 test1: 0.8790383        best: 0.8790383 (300)   tota
l: 18.5s        remaining: 43s
310:    test: 0.8899169 test1: 0.8792089        best: 0.8792089 (310)   tota
l: 19.1s        remaining: 42.2s
320:    test: 0.8902279 test1: 0.8795504        best: 0.8795504 (320)   tota
l: 19.6s        remaining: 41.4s
330:    test: 0.8908921 test1: 0.8803104        best: 0.8803115 (329)   tota
l: 20.1s        remaining: 40.7s
340:    test: 0.8913042 test1: 0.8807369        best: 0.8807369 (340)   tota
l: 20.7s        remaining: 40s
350:    test: 0.8917350 test1: 0.8810404        best: 0.8810404 (350)   tota
l: 21.2s        remaining: 39.3s
360:    test: 0.8919213 test1: 0.8812479        best: 0.8812479 (360)   tota
l: 21.8s        remaining: 38.5s
370:    test: 0.8922800 test1: 0.8814699        best: 0.8814919 (369)   tota
l: 22.3s        remaining: 37.8s
380:    test: 0.8926446 test1: 0.8817711        best: 0.8817773 (378)   tota
l: 22.8s        remaining: 37.1s
390:    test: 0.8930844 test1: 0.8820645        best: 0.8820645 (390)   tota
l: 23.4s        remaining: 36.4s
400:    test: 0.8935259 test1: 0.8824410        best: 0.8824410 (400)   tota
l: 23.9s        remaining: 35.7s
410:    test: 0.8937299 test1: 0.8826171        best: 0.8826171 (410)   tota
l: 24.5s        remaining: 35s
420:    test: 0.8938340 test1: 0.8827120        best: 0.8827161 (419)   tota
l: 25s  remaining: 34.4s
430:    test: 0.8938414 test1: 0.8827093        best: 0.8827161 (419)   tota
l: 25.5s        remaining: 33.6s
440:    test: 0.8938376 test1: 0.8826936        best: 0.8827161 (419)   tota
l: 26.1s        remaining: 33s
450:    test: 0.8938400 test1: 0.8826844        best: 0.8827161 (419)   tota
l: 26.8s        remaining: 32.6s
460:    test: 0.8938429 test1: 0.8826791        best: 0.8827161 (419)   tota
l: 27.5s        remaining: 32.1s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8827161236
bestIteration = 419

Shrink model to first 420 iterations.
```

Out[26]:

```
<catboost.core.CatBoostClassifier at 0xf854b60e20>
```

В [27]:

```
train_scores["CatBoost_num_features"] = cb_model.predict_proba(x_train[xgb_numerical_featur
test_scores["CatBoost_num_features"] = cb_model.predict_proba(x_test[xgb_numerical_features
```

В [28]:

```
y_pred = cb_model.predict_proba(df_test_new[xgb_numerical_features])[:,1]
```

В [29]:

```
score = roc_auc_score(df_test_new["isFraud"], y_pred)
score
```

Out[29]:

0.8513559235540431

Задание 1:

Признак TransactionDT - это смещение в секундах относительно базовой даты. Базовая дата - 2017-12-01, преобразовать признак TransactionDT в datetime, прибавив к базовой дате исходное значение признака. Из полученного признака выделить год, месяц, день недели, час, день.

CatBoost на числовых признаках

В [30]:

```
import datetime
```

В [31]:

```
# Значение: datetime.datetime(2017, 4, 5, 0, 18, 51, 980187)
# now = datetime.datetime.now()
# base_date = datetime.datetime(2017, 10, 1)
# d = datetime.timedelta(seconds=11316)
# date = base_date + d
# print(now)
# print(date)
# print(date.year)
# print(date.month)
# print(date.day)
# print(date.hour)
# print(date.minute)
# print(date.second)
# print(date.weekday())
```

В [32]:

```
# def function(x):
#     return datetime.timedelta(seconds=x)

# df = pd.DataFrame({'TransactionDT': [86400, 86401, 86402]})
# df['DT'] = df['TransactionDT'].apply(function)
# df
```

В [33]:

```
def function(x):
    base_date = datetime.datetime(2017, 10, 1)
    new_date = base_date + datetime.timedelta(seconds=x)
    year = new_date.year
    month = new_date.month
    week_day = new_date.weekday()
    hour = new_date.hour
    day = new_date.day
    #return new_date, year, month, week_day, hour, day
    return year, month, week_day, hour, day

# df['new_date'], df['year'], df['month'], df['week_day'], df['hour'], df['day']  = zip(*df
# df
```

В [34]:

```
x_train_task_1 = x_train[xgb_numerical_features + catigorical_features].copy()
x_test_task_1 = x_test[xgb_numerical_features + catigorical_features].copy()
#df_test_new_task_1 = df_test_new[['TransactionID', 'isFraud'] + xgb_numerical_features].co
df_test_new_task_1 = df_test_new[['isFraud'] + xgb_numerical_features + catigorical_feature

# x_train_task_1['new_date'],
x_train_task_1['year'], x_train_task_1['month'], x_train_task_1['week_day'], x_train_task_1
zip(*x_train_task_1['TransactionDT'].map(function))

# x_test_task_1['new_date'],
x_test_task_1['year'], x_test_task_1['month'], x_test_task_1['week_day'], x_test_task_1['ho
zip(*x_test_task_1['TransactionDT'].map(function))
```

В [35]:

```
df_test_new_task_1['year'], df_test_new_task_1['month'], df_test_new_task_1['week_day'], df
zip(*df_test_new_task_1['TransactionDT'].map(function))

#x_train_task_1.columns
```

В [36]:

```
task_1_fields = ['year', 'month', 'week_day', 'hour', 'day']
```

В [37]:

```
eval_sets= [
    (x_train_task_1[xgb_numerical_features + task_1_fields], y_train),
    (x_test_task_1[xgb_numerical_features + task_1_fields], y_test)
]
```

В [38]:

```
cb_model = cb.CatBoostClassifier(**cb_params)
cb_model.fit(x_train_task_1[xgb_numerical_features + task_1_fields], y_train, eval_set=eval
```

```
0:	test: 0.6667114 test1: 0.6618119	best: 0.6618119 (0)	tota
l: 198ms	remaining: 3m 18s
10:	test: 0.7583015 test1: 0.7459275	best: 0.7459275 (10)	tota
l: 1.28s	remaining: 1m 55s
20:	test: 0.8245631 test1: 0.8100912	best: 0.8100912 (20)	tota
l: 2.12s	remaining: 1m 38s
30:	test: 0.8459280 test1: 0.8345029	best: 0.8345029 (30)	tota
l: 2.76s	remaining: 1m 26s
40:	test: 0.8517896 test1: 0.8401610	best: 0.8401610 (40)	tota
l: 3.31s	remaining: 1m 17s
50:	test: 0.8557151 test1: 0.8445617	best: 0.8445617 (50)	tota
l: 3.87s	remaining: 1m 12s
60:	test: 0.8568242 test1: 0.8454821	best: 0.8459765 (57)	tota
l: 4.44s	remaining: 1m 8s
70:	test: 0.8597014 test1: 0.8481595	best: 0.8481595 (70)	tota
l: 5.01s	remaining: 1m 5s
80:	test: 0.8633124 test1: 0.8520402	best: 0.8521644 (79)	tota
l: 5.61s	remaining: 1m 3s
90:	test: 0.8661622 test1: 0.8550718	best: 0.8550718 (90)	tota
l: 6.18s	remaining: 1m 1s
100:	test: 0.8678117 test1: 0.8566115	best: 0.8566115 (100)	tota
l: 6.74s	remaining: 60s
110:	test: 0.8693647 test1: 0.8588993	best: 0.8588993 (110)	tota
l: 7.34s	remaining: 58.8s
120:	test: 0.8701545 test1: 0.8594978	best: 0.8594978 (120)	tota
l: 7.92s	remaining: 57.6s
130:	test: 0.8710036 test1: 0.8603732	best: 0.8603732 (130)	tota
l: 8.5s remaining: 56.4s
140:	test: 0.8727777 test1: 0.8620637	best: 0.8620637 (140)	tota
l: 9.22s	remaining: 56.2s
150:	test: 0.8752387 test1: 0.8648728	best: 0.8648728 (150)	tota
l: 9.8s remaining: 55.1s
160:	test: 0.8772954 test1: 0.8666369	best: 0.8666369 (160)	tota
l: 10.4s	remaining: 54.2s
170:	test: 0.8796656 test1: 0.8691725	best: 0.8691725 (170)	tota
l: 11s	remaining: 53.2s
180:	test: 0.8809885 test1: 0.8705277	best: 0.8705277 (180)	tota
l: 11.6s	remaining: 52.4s
190:	test: 0.8819119 test1: 0.8711312	best: 0.8711312 (190)	tota
l: 12.2s	remaining: 51.7s
200:	test: 0.8831271 test1: 0.8720865	best: 0.8720865 (200)	tota
l: 12.9s	remaining: 51.2s
210:	test: 0.8837480 test1: 0.8727085	best: 0.8727085 (210)	tota
l: 13.4s	remaining: 50.2s
220:	test: 0.8847731 test1: 0.8738636	best: 0.8738636 (220)	tota
l: 14s	remaining: 49.4s
230:	test: 0.8859376 test1: 0.8749963	best: 0.8750045 (229)	tota
l: 14.6s	remaining: 48.5s
240:	test: 0.8865317 test1: 0.8753854	best: 0.8754155 (238)	tota
l: 15.1s	remaining: 47.7s
250:	test: 0.8872459 test1: 0.8758372	best: 0.8758372 (250)	tota
l: 15.7s	remaining: 46.9s
260:	test: 0.8876787 test1: 0.8761922	best: 0.8761922 (260)	tota
l: 16.3s	remaining: 46.2s
270:	test: 0.8879844 test1: 0.8763156	best: 0.8763220 (269)	tota
l: 16.9s	remaining: 45.5s
```

```
280:	test: 0.8887235 test1: 0.8769977	best: 0.8769977 (280)	tota
l: 17.5s	remaining: 44.7s
290:	test: 0.8893078 test1: 0.8773692	best: 0.8773823 (288)	tota
l: 18s	remaining: 43.9s
300:	test: 0.8898077 test1: 0.8779091	best: 0.8779091 (300)	tota
l: 18.6s	remaining: 43.1s
310:	test: 0.8903706 test1: 0.8785188	best: 0.8785242 (309)	tota
l: 19.1s	remaining: 42.3s
320:	test: 0.8907192 test1: 0.8787738	best: 0.8787738 (320)	tota
l: 19.6s	remaining: 41.5s
330:	test: 0.8912989 test1: 0.8792003	best: 0.8792003 (330)	tota
l: 20.2s	remaining: 40.8s
340:	test: 0.8917896 test1: 0.8795784	best: 0.8795784 (340)	tota
l: 20.7s	remaining: 40.1s
350:	test: 0.8922154 test1: 0.8798731	best: 0.8798731 (350)	tota
l: 21.3s	remaining: 39.4s
360:	test: 0.8924780 test1: 0.8800431	best: 0.8800431 (360)	tota
l: 21.8s	remaining: 38.6s
370:	test: 0.8927055 test1: 0.8802442	best: 0.8802442 (370)	tota
l: 22.4s	remaining: 37.9s
380:	test: 0.8931425 test1: 0.8806171	best: 0.8806171 (380)	tota
l: 22.9s	remaining: 37.2s
390:	test: 0.8934096 test1: 0.8809053	best: 0.8809053 (390)	tota
l: 23.4s	remaining: 36.5s
400:	test: 0.8936037 test1: 0.8810861	best: 0.8810861 (400)	tota
l: 23.9s	remaining: 35.8s
410:	test: 0.8937058 test1: 0.8811599	best: 0.8811599 (410)	tota
l: 24.4s	remaining: 35s
420:	test: 0.8938017 test1: 0.8812334	best: 0.8812334 (420)	tota
l: 24.9s	remaining: 34.3s
430:	test: 0.8938038 test1: 0.8812265	best: 0.8812334 (420)	tota
l: 25.4s	remaining: 33.6s
440:	test: 0.8938145 test1: 0.8812315	best: 0.8812334 (420)	tota
l: 25.9s	remaining: 32.9s
450:	test: 0.8938250 test1: 0.8812374	best: 0.8812377 (449)	tota
l: 26.5s	remaining: 32.2s
460:	test: 0.8938320 test1: 0.8812370	best: 0.8812417 (455)	tota
l: 27s	remaining: 31.5s
470:	test: 0.8938257 test1: 0.8812249	best: 0.8812417 (455)	tota
l: 27.5s	remaining: 30.9s
480:	test: 0.8938242 test1: 0.8812157	best: 0.8812417 (455)	tota
l: 28s	remaining: 30.2s
490:	test: 0.8938261 test1: 0.8812111	best: 0.8812417 (455)	tota
l: 28.5s	remaining: 29.5s
500:	test: 0.8938288 test1: 0.8812090	best: 0.8812417 (455)	tota
l: 29s	remaining: 28.9s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8812417137
bestIteration = 455

Shrink model to first 456 iterations.
```

Out[38]:

```
<catboost.core.CatBoostClassifier at 0xf85444f550>
```

Задание 0:

- bestTest = 0.8827161236
- bestIteration = 419

Задание 1:

- bestTest = 0.8812417137
- bestIteration = 455

Вывод:

- Добавление новых признаков (Задание 1) не дало улучшения качества модели.

В [39]:

```
train_scores["CatBoost_task1_features"] = cb_model.predict_proba(x_train_task_1[xgb_numeric
test_scores["CatBoost_task1_features"] = cb_model.predict_proba(x_test_task_1[xgb_numerical
#train_scores["CatBoost_num_features"] = cb_model.predict_proba(x_train_task_1[xgb_numerica
#test_scores["CatBoost_num_features"] = cb_model.predict_proba(x_test_task_1[xgb_numerical_
```

В [40]:

```
y_pred = cb_model.predict_proba(df_test_new_task_1[['isFraud'] + xgb_numerical_features + t
```

В [41]:

```
score = roc_auc_score(df_test_new_task_1["isFraud"], y_pred)
score
```

Out[41]:

```
0.8541448109129632
```

Задание 0:

- 0.8513559235540431

Задание 1:

- 0.8541448109129632

Вывод:

- Добавление новых признаков улучшило качество модели.

Задание 2:

Сделать конкатенацию признаков

- card1 + card2;
- card1 + card2 + card_3 + card_5;
- card1 + card2 + card_3 + card_5 + addr1 + addr2

Рассматривать их как категориальных признаки.

В [42]:

```
# import pandas as pd
# df = pd.DataFrame({'foo':['a','b','c'], 'bar':[1, 2, 3]})
# df['baz'] = df.agg(lambda x: f"{x['bar']} is {x['foo']}", axis=1)
# df
```

В [43]:

```
x_train_task_1.columns
```

Out[43]:

```
Index(['TransactionDT', 'TransactionAmt', 'card1', 'card2', 'card3', 'card
5', 'addr1', 'addr2', 'C1', 'C2',
       ...
       'M5', 'M6', 'M7', 'M8', 'M9', 'year', 'month', 'week_day', 'hour', 'd
ay'], dtype='object', length=213)
```

В [44]:

```
# x_train_task_1['card1_card2'] = x_train_task_1.agg(lambda x: x['card1'] + x['card2'], axi
# x_train_task_1['card1_card2_card_3_card_5'] = \
#         x_train_task_1.agg(lambda x: x['card1_card2'] + x['card5'] + x['card5'], axis=1)
# x_train_task_1['card1_card2_card_3_card_5_addr1_addr2'] = \
#             x_train_task_1.agg(lambda x: f"{x['card1_card2_card_3_card_5']} + {x['addr1']

x_train_task_1['card1_card2'] = x_train_task_1.agg(lambda x: f"{x['card1']} {x['card2']}",
x_train_task_1['card1_card2_card_3_card_5'] = \
    x_train_task_1.agg(lambda x: f"{x['card1_card2']} {x['card3']} {x['card5']}", axis=1)
x_train_task_1['card1_card2_card_3_card_5_addr1_addr2'] = \
    x_train_task_1.agg(lambda x: f"{x['card1_card2_card_3_card_5']} {x['addr1']} {x['addr2'

# x_train_task_1[['card1_card2', 'card1_card2_card_3_card_5', 'card1_card2_card_3_card_5_ad
```

В [45]:

```
x_train_task_1[['card1_card2', 'card1_card2_card_3_card_5', 'card1_card2_card_3_card_5_addr
```

Out[45]:

	card1_card2	card1_card2_card_3_card_5	card1_card2_card_3_card_5_addr1_addr2	year
141582	6892 560.0	6892 560.0 150.0 226.0	6892 560.0 150.0 226.0 433.0 87.0	2017
131503	2922 583.0	2922 583.0 150.0 226.0	2922 583.0 150.0 226.0 299.0 87.0	2017

В [46]:

```
x_test_task_1['card1_card2'] = x_test_task_1.agg(lambda x: f"{x['card1']} {x['card2']}", ax
x_test_task_1['card1_card2_card_3_card_5'] = \
    x_test_task_1.agg(lambda x: f"{x['card1_card2']} {x['card3']} {x['card5']}", axis=1)
x_test_task_1['card1_card2_card_3_card_5_addr1_addr2'] = \
    x_test_task_1.agg(lambda x: f"{x['card1_card2_card_3_card_5']} {x['addr1']} {x['addr2']
```

В [47]:

```
x_test_task_1[['card1_card2', 'card1_card2_card_3_card_5', 'card1_card2_card_3_card_5_addr1
```

Out[47]:

	card1_card2	card1_card2_card_3_card_5	card1_card2_card_3_card_5_addr1_addr2	year
78715	15186 480.0	15186 480.0 150.0 224.0	15186 480.0 150.0 224.0 299.0 87.0	2017
907	6019 583.0	6019 583.0 150.0 226.0	6019 583.0 150.0 226.0 225.0 87.0	2017

В [48]:

```
x_test_task_1['card1_card2'] = x_test_task_1.agg(lambda x: f"{x['card1']} {x['card2']}", ax
x_test_task_1['card1_card2_card_3_card_5'] = \
    x_test_task_1.agg(lambda x: f"{x['card1_card2']} {x['card3']} {x['card5']}", axis=1)
x_test_task_1['card1_card2_card_3_card_5_addr1_addr2'] = \
    x_test_task_1.agg(lambda x: f"{x['card1_card2_card_3_card_5']} {x['addr1']} {x['addr2']
```

В [49]:

```
# x_train_task_1.info()
categorical_features = x_train_task_1.select_dtypes(include=["object"]).columns
x_train_task_1[categorical_features] = x_train_task_1[categorical_features].astype(str)
x_test_task_1[categorical_features] = x_test_task_1[categorical_features].astype(str)
#categorical_features = []
categorical_features = ['card1_card2', 'card1_card2_card_3_card_5', 'card1_card2_card_3_car
categorical_features
```

Out[49]:

```
['card1_card2',
 'card1_card2_card_3_card_5',
 'card1_card2_card_3_card_5_addr1_addr2']
```

В [50]:

```
# x_test_task_1 = x_test[xgb_numerical_features].copy()
# df_test_new_task_1 = df_test_new[['TransactionID', 'isFraud'] + xgb_numerical_features].c
# df_test_new_task_1 = df_test_new[['isFraud'] + xgb_numerical_features].copy()
```

CatBoost с категориальными признаками

В [51]:

```
# eval_sets= [
#     (x_train_task_1[xgb_numerical_features + task_1_fields + categorical_features], y_tra
#     (x_test_task_1[xgb_numerical_features + task_1_fields + categorical_features], y_test
# ]
eval_sets= [
    (x_train_task_1[xgb_numerical_features + categorical_features], y_train),
    (x_test_task_1[xgb_numerical_features + categorical_features], y_test)
]
```

В [52]:

```
# cb_model.fit(
#     x_train_task_1[xgb_numerical_features + task_1_fields + categorical_features],
#     y_train,
#     cat_features = categorical_features,
#     eval_set=eval_sets)
cb_model.fit(
    x_train_task_1[xgb_numerical_features + categorical_features],
    y_train,
    cat_features = categorical_features,
    eval_set=eval_sets)
```

```
0:	test: 0.6169405 test1: 0.6013935	best: 0.6013935 (0)	to
tal: 431ms	remaining: 7m 10s
10:	test: 0.7872496 test1: 0.7697118	best: 0.7697118 (10)	to
tal: 3.65s	remaining: 5m 27s
20:	test: 0.8188597 test1: 0.8034291	best: 0.8034291 (20)	to
tal: 6.46s	remaining: 5m 1s
30:	test: 0.8414185 test1: 0.8284577	best: 0.8284577 (30)	to
tal: 9.37s	remaining: 4m 53s
40:	test: 0.8625035 test1: 0.8473969	best: 0.8478863 (39)	to
tal: 12.4s	remaining: 4m 50s
50:	test: 0.9180574 test1: 0.8809312	best: 0.8809312 (50)	to
tal: 15s	remaining: 4m 38s
60:	test: 0.9245797 test1: 0.8846792	best: 0.8846792 (60)	to
tal: 18.1s	remaining: 4m 38s
70:	test: 0.9267131 test1: 0.8856667	best: 0.8857395 (68)	to
tal: 20.2s	remaining: 4m 23s
80:	test: 0.9278434 test1: 0.8857564	best: 0.8859937 (76)	to
tal: 22.5s	remaining: 4m 15s
90:	test: 0.9290020 test1: 0.8884479	best: 0.8884479 (90)	to
tal: 25s	remaining: 4m 9s
100:	test: 0.9299823 test1: 0.8911254	best: 0.8911254 (100)	to
tal: 27.9s	remaining: 4m 8s
110:	test: 0.9317724 test1: 0.8940895	best: 0.8940895 (110)	to
tal: 30.5s	remaining: 4m 4s
120:	test: 0.9327979 test1: 0.8957085	best: 0.8957085 (120)	to
tal: 33.3s	remaining: 4m 1s
130:	test: 0.9329699 test1: 0.8963070	best: 0.8963070 (130)	to
tal: 35.7s	remaining: 3m 56s
140:	test: 0.9333574 test1: 0.8966002	best: 0.8966983 (139)	to
tal: 38.5s	remaining: 3m 54s
150:	test: 0.9341148 test1: 0.8972911	best: 0.8972911 (150)	to
tal: 41.2s	remaining: 3m 51s
160:	test: 0.9358735 test1: 0.8985740	best: 0.8985740 (160)	to
tal: 44s	remaining: 3m 49s
170:	test: 0.9369014 test1: 0.9001801	best: 0.9001801 (170)	to
tal: 46.8s	remaining: 3m 46s
180:	test: 0.9376979 test1: 0.9016967	best: 0.9016967 (180)	to
tal: 50s	remaining: 3m 46s
190:	test: 0.9384334 test1: 0.9030432	best: 0.9030432 (190)	to
tal: 52.5s	remaining: 3m 42s
200:	test: 0.9388313 test1: 0.9037380	best: 0.9037392 (199)	to
tal: 55.4s	remaining: 3m 40s
210:	test: 0.9393869 test1: 0.9049347	best: 0.9049557 (208)	to
tal: 58.2s	remaining: 3m 37s
220:	test: 0.9399397 test1: 0.9058418	best: 0.9058418 (220)	to
tal: 1m 1s	remaining: 3m 37s
230:	test: 0.9405147 test1: 0.9066829	best: 0.9066829 (230)	to
```

```
tal: 1m 4s      remaining: 3m 34s
240:    test: 0.9408963 test1: 0.9073364        best: 0.9073364 (240)   to
tal: 1m 6s      remaining: 3m 28s
250:    test: 0.9415587 test1: 0.9082409        best: 0.9082409 (250)   to
tal: 1m 9s      remaining: 3m 26s
260:    test: 0.9422290 test1: 0.9091341        best: 0.9091341 (260)   to
tal: 1m 12s     remaining: 3m 25s
270:    test: 0.9425660 test1: 0.9095927        best: 0.9095927 (270)   to
tal: 1m 15s     remaining: 3m 23s
280:    test: 0.9430781 test1: 0.9101844        best: 0.9101844 (280)   to
tal: 1m 18s     remaining: 3m 19s
290:    test: 0.9434543 test1: 0.9107689        best: 0.9107689 (290)   to
tal: 1m 21s     remaining: 3m 18s
300:    test: 0.9437471 test1: 0.9112131        best: 0.9112154 (298)   to
tal: 1m 24s     remaining: 3m 17s
310:    test: 0.9441455 test1: 0.9117472        best: 0.9117472 (310)   to
tal: 1m 29s     remaining: 3m 17s
320:    test: 0.9451149 test1: 0.9127553        best: 0.9127553 (320)   to
tal: 1m 32s     remaining: 3m 15s
330:    test: 0.9452018 test1: 0.9130111        best: 0.9130114 (329)   to
tal: 1m 35s     remaining: 3m 13s
340:    test: 0.9457598 test1: 0.9133788        best: 0.9133788 (340)   to
tal: 1m 39s     remaining: 3m 12s
350:    test: 0.9458491 test1: 0.9135031        best: 0.9135031 (350)   to
tal: 1m 42s     remaining: 3m 9s
360:    test: 0.9468890 test1: 0.9144436        best: 0.9144436 (360)   to
tal: 1m 45s     remaining: 3m 6s
370:    test: 0.9471215 test1: 0.9148807        best: 0.9148807 (370)   to
tal: 1m 48s     remaining: 3m 3s
380:    test: 0.9482912 test1: 0.9159234        best: 0.9159234 (380)   to
tal: 1m 50s     remaining: 2m 59s
390:    test: 0.9483800 test1: 0.9161118        best: 0.9161136 (388)   to
tal: 1m 53s     remaining: 2m 56s
400:    test: 0.9497767 test1: 0.9173843        best: 0.9173843 (400)   to
tal: 1m 56s     remaining: 2m 54s
410:    test: 0.9508740 test1: 0.9184315        best: 0.9184315 (410)   to
tal: 1m 58s     remaining: 2m 50s
420:    test: 0.9512871 test1: 0.9188090        best: 0.9188090 (420)   to
tal: 2m 1s      remaining: 2m 47s
430:    test: 0.9515353 test1: 0.9190670        best: 0.9190670 (430)   to
tal: 2m 4s      remaining: 2m 43s
440:    test: 0.9525292 test1: 0.9199369        best: 0.9199369 (440)   to
tal: 2m 7s      remaining: 2m 41s
450:    test: 0.9525703 test1: 0.9199884        best: 0.9199884 (449)   to
tal: 2m 10s     remaining: 2m 38s
460:    test: 0.9526155 test1: 0.9199909        best: 0.9199909 (460)   to
tal: 2m 12s     remaining: 2m 34s
470:    test: 0.9526270 test1: 0.9199953        best: 0.9199972 (467)   to
tal: 2m 14s     remaining: 2m 31s
480:    test: 0.9531521 test1: 0.9205022        best: 0.9205022 (479)   to
tal: 2m 18s     remaining: 2m 29s
490:    test: 0.9531602 test1: 0.9204948        best: 0.9205032 (481)   to
tal: 2m 19s     remaining: 2m 24s
500:    test: 0.9531621 test1: 0.9205005        best: 0.9205033 (497)   to
tal: 2m 21s     remaining: 2m 20s
510:    test: 0.9531664 test1: 0.9204961        best: 0.9205033 (497)   to
tal: 2m 24s     remaining: 2m 18s
520:    test: 0.9531864 test1: 0.9204943        best: 0.9205033 (497)   to
tal: 2m 26s     remaining: 2m 14s
530:    test: 0.9531969 test1: 0.9205009        best: 0.9205033 (497)   to
tal: 2m 27s     remaining: 2m 10s
```

```
540:    test: 0.9531981 test1: 0.9204980        best: 0.9205033 (497)   to
tal: 2m 29s     remaining: 2m 6s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.9205033376
bestIteration = 497

Shrink model to first 498 iterations.
```

Out[52]:

```
<catboost.core.CatBoostClassifier at 0xf85444f550>
```

Задание 0:

- bestTest = 0.8827161236
- bestIteration = 419

Задание 2:

- bestTest = 0.9205033376
- bestIteration = 497

Вывод:

- Добавление новых признаков (Задание 2) значительно улучшило качество модели по сравнению с базовым решением.

В [54]:

```
# train_scores["CatBoost_task2_features"] = \
#     cb_model.predict_proba(x_train_task_1[xgb_numerical_features + task_1_fields + catego
train_scores["CatBoost_task2_features"] = \
    cb_model.predict_proba(x_train_task_1[xgb_numerical_features + categorical_features])[:
```

В [55]:

```
# test_scores["CatBoost_task2_features"] = \
#     cb_model.predict_proba(x_test_task_1[xgb_numerical_features + task_1_fields + categor
test_scores["CatBoost_task2_features"] = \
    cb_model.predict_proba(x_test_task_1[xgb_numerical_features + categorical_features])[:,
```

Задание 3:

Сделать *Frequency Encoding* для признаков *card1* - *card6, addr1, addr2*.

В [56]:

```
data = []
data_test = []
data = x_train_task_1.copy()
data_test = x_test_task_1.copy()
```

В [57]:

```
freq_encoder = data["card1"].value_counts(normalize=True)
data["card1_freq_enc"] = data["card1"].map(freq_encoder)
freq_encoder = data["card2"].value_counts(normalize=True)
data["card2_freq_enc"] = data["card2"].map(freq_encoder)
freq_encoder = data["card3"].value_counts(normalize=True)
data["card3_freq_enc"] = data["card3"].map(freq_encoder)
freq_encoder = data["card4"].value_counts(normalize=True)
data["card4_freq_enc"] = data["card4"].map(freq_encoder)
freq_encoder = data["card5"].value_counts(normalize=True)
data["card5_freq_enc"] = data["card5"].map(freq_encoder)
freq_encoder = data["card6"].value_counts(normalize=True)
data["card6_freq_enc"] = data["card6"].map(freq_encoder)
freq_encoder = data["addr1"].value_counts(normalize=True)
data["addr1_freq_enc"] = data["addr1"].map(freq_encoder)
freq_encoder = data["addr2"].value_counts(normalize=True)
data["addr2_freq_enc"] = data["addr2"].map(freq_encoder)
# https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02
# fe = data.groupby('card1').size()/len(data)
# data.loc[:, 'card1_freq_enc'] = data['card1'].map(fe)
# fe = data.groupby('card2').size()/len(data)
# data.loc[:, 'card2_freq_enc'] = data['card2'].map(fe)
# fe = data.groupby('card3').size()/len(data)
# data.loc[:, 'card3_freq_enc'] = data['card3'].map(fe)
# fe = data.groupby('card4').size()/len(data)
# data.loc[:, 'card4_freq_enc'] = data['card4'].map(fe)
# fe = data.groupby('card5').size()/len(data)
# data.loc[:, 'card5_freq_enc'] = data['card5'].map(fe)
# fe = data.groupby('card6').size()/len(data)
# data.loc[:, 'card6_freq_enc'] = data['card6'].map(fe)
# fe = data.groupby('addr1').size()/len(data)
# data.loc[:, 'addr1_freq_enc'] = data['addr1'].map(fe)
# fe = data.groupby('addr2').size()/len(data)
# data.loc[:, 'addr2_freq_enc'] = data['addr2'].map(fe)
```

В [58]:

```
data[['card1', 'card1_freq_enc', 'card2', 'card2_freq_enc', 'card3', 'card3_freq_enc', \
     'card4', 'card4_freq_enc', 'card5', 'card5_freq_enc', 'card6', 'card6_freq_enc', \
     'addr1', 'addr1_freq_enc', 'addr2', 'addr2_freq_enc']].head(2)
# Функция map применяет функцию к каждому элементу последовательности и возвращает итератор
```

Out[58]:

	card1	card1_freq_enc	card2	card2_freq_enc	card3	card3_freq_enc	card4	card4_fre
141582	6892	0.000311	560.0	0.000436	150.0	0.879139	visa	0.6
131503	2922	0.000104	583.0	0.054646	150.0	0.879139	visa	0.6

В [59]:

```
# https://towardsdatascience.com/all-about-categorical-variable-encoding-305f3361fd02
# fe = data_test.groupby('card1').size()/len(data_test)
# data_test.loc[:, 'card1_freq_encode'] = data_test['card1'].map(fe)
# fe = data_test.groupby('card2').size()/len(data_test)
# data_test.loc[:, 'card2_freq_encode'] = data_test['card2'].map(fe)
# fe = data_test.groupby('card3').size()/len(data_test)
# data_test.loc[:, 'card3_freq_encode'] = data_test['card3'].map(fe)
# fe = data_test.groupby('card4').size()/len(data_test)
# data_test.loc[:, 'card4_freq_encode'] = data_test['card4'].map(fe)
# fe = data_test.groupby('card5').size()/len(data_test)
# data_test.loc[:, 'card5_freq_encode'] = data_test['card5'].map(fe)
# fe = data_test.groupby('card6').size()/len(data_test)
# data_test.loc[:, 'card6_freq_encode'] = data_test['card6'].map(fe)
# fe = data_test.groupby('addr1').size()/len(data_test)
# data_test.loc[:, 'addr1_freq_encode'] = data_test['addr1'].map(fe)
# fe = data_test.groupby('addr2').size()/len(data_test)
# data_test.loc[:, 'addr2_freq_encode'] = data_test['addr2'].map(fe)
freq_encoder = data_test["card1"].value_counts(normalize=True)
data_test["card1_freq_enc"] = data_test["card1"].map(freq_encoder)
freq_encoder = data_test["card2"].value_counts(normalize=True)
data_test["card2_freq_enc"] = data_test["card2"].map(freq_encoder)
freq_encoder = data_test["card3"].value_counts(normalize=True)
data_test["card3_freq_enc"] = data_test["card3"].map(freq_encoder)
freq_encoder = data_test["card4"].value_counts(normalize=True)
data_test["card4_freq_enc"] = data_test["card4"].map(freq_encoder)
freq_encoder = data_test["card5"].value_counts(normalize=True)
data_test["card5_freq_enc"] = data_test["card5"].map(freq_encoder)
freq_encoder = data_test["card6"].value_counts(normalize=True)
data_test["card6_freq_enc"] = data_test["card6"].map(freq_encoder)
freq_encoder = data_test["addr1"].value_counts(normalize=True)
data_test["addr1_freq_enc"] = data_test["addr1"].map(freq_encoder)
freq_encoder = data_test["addr2"].value_counts(normalize=True)
data_test["addr2_freq_enc"] = data_test["addr2"].map(freq_encoder)
```

В [60]:

```
data_test[['card1', 'card1_freq_enc', 'card2', 'card2_freq_enc', 'card3', 'card3_freq_enc',
     'card4', 'card4_freq_enc', 'card5', 'card5_freq_enc', 'card6', 'card6_freq_enc', \
     'addr1', 'addr1_freq_enc', 'addr2', 'addr2_freq_enc']].head(2)
# Функция map применяет функцию к каждому элементу последовательности и возвращает итератор
```

Out[60]:

	card1	card1_freq_enc	card2	card2_freq_enc	card3	card3_freq_enc	card4	card4_
78715	15186	0.000267	480.0	0.003451	150.0	0.881531	mastercard	
907	6019	0.018267	583.0	0.055197	150.0	0.881531	visa	

В [61]:

```
# task3_cat_features = ['card1_freq_encode', 'card2_freq_encode', 'card3_freq_encode', \
#       'card4_freq_encode', 'card5_freq_encode', 'card6_freq_encode', 'addr1_freq_encode',
# categorical_features = categorical_features + task3_cat_features
categorical_features = ['card1_card2',
 'card1_card2_card_3_card_5',
 'card1_card2_card_3_card_5_addr1_addr2',
 'card1_freq_enc',
 'card2_freq_enc',
 'card3_freq_enc',
 'card4_freq_enc',
 'card5_freq_enc',
 'card6_freq_enc',
 'addr1_freq_enc',
 'addr2_freq_enc',
 'card4',
 'card6'
]
#categorical_features = x_train_task_3.select_dtypes(include=["object"]).columns
```

В [62]:

```
x_train_task_3 = data[xgb_numerical_features + task_1_fields + categorical_features].copy()
```

В [63]:

```
x_train_task_3["card4"].head(2)
```

Out[63]:

```
141582    visa
131503    visa
Name: card4, dtype: category
Categories (5, object): ['american express', 'discover', 'mastercard', 'vis
a', 'Unknown']
```

В [64]:

```
x_train_task_3[categorical_features] = x_train_task_3[categorical_features].astype(str)
```

В [65]:

```
x_test_task_3 = data_test[xgb_numerical_features + task_1_fields + categorical_features].co
x_test_task_3[categorical_features] = x_test_task_3[categorical_features].astype(str)
```

В [66]:

```
#x_test_task_3.isnull().sum(axis = 0)
```

B [67]:

```
# eval_sets= [
#     (x_train_task_3[xgb_numerical_features + task_1_fields + categorical_features], y_tra
#     (x_test_task_3[xgb_numerical_features + task_1_fields + categorical_features], y_test
# ]
eval_sets= [
    (x_train_task_3[xgb_numerical_features + categorical_features], y_train),
    (x_test_task_3[xgb_numerical_features + categorical_features], y_test)
]
```

В [68]:

```
# cb_model.fit(
#     x_train_task_3[xgb_numerical_features + task_1_fields + categorical_features],
#     y_train,
#     cat_features = categorical_features,
#     eval_set=eval_sets)

cb_model.fit(
    x_train_task_3[xgb_numerical_features + categorical_features],
    y_train,
    cat_features = categorical_features,
    eval_set=eval_sets)
```

```
0:	test: 0.6495082 test1: 0.4114233	best: 0.4114233 (0)	tota
l: 781ms	remaining: 13m
10:	test: 0.7880146 test1: 0.7277898	best: 0.7584509 (8)	tota
l: 4.79s	remaining: 7m 10s
20:	test: 0.8285493 test1: 0.8307867	best: 0.8307867 (20)	tota
l: 7.66s	remaining: 5m 56s
30:	test: 0.8470180 test1: 0.8354330	best: 0.8380251 (28)	tota
l: 10.8s	remaining: 5m 37s
40:	test: 0.8576627 test1: 0.8261190	best: 0.8380251 (28)	tota
l: 14.2s	remaining: 5m 32s
50:	test: 0.8862479 test1: 0.8623361	best: 0.8623361 (50)	tota
l: 17.1s	remaining: 5m 19s
60:	test: 0.9139052 test1: 0.8819011	best: 0.8819011 (60)	tota
l: 20s	remaining: 5m 8s
70:	test: 0.9222446 test1: 0.8862534	best: 0.8862534 (70)	tota
l: 23.4s	remaining: 5m 6s
80:	test: 0.9252371 test1: 0.8875602	best: 0.8875602 (80)	tota
l: 26.5s	remaining: 5m
90:	test: 0.9272170 test1: 0.8892153	best: 0.8893226 (88)	tota
l: 29.3s	remaining: 4m 52s
100:	test: 0.9287967 test1: 0.8920581	best: 0.8920581 (100)	tota
l: 32.4s	remaining: 4m 48s
110:	test: 0.9298694 test1: 0.8937770	best: 0.8937770 (110)	tota
l: 35.2s	remaining: 4m 42s
120:	test: 0.9309668 test1: 0.8952842	best: 0.8952842 (120)	tota
l: 38.3s	remaining: 4m 38s
130:	test: 0.9322053 test1: 0.8971725	best: 0.8971745 (129)	tota
l: 41.7s	remaining: 4m 36s
140:	test: 0.9323883 test1: 0.8973558	best: 0.8973558 (140)	tota
l: 44.6s	remaining: 4m 31s
150:	test: 0.9325430 test1: 0.8977638	best: 0.8977638 (150)	tota
l: 47.7s	remaining: 4m 28s
160:	test: 0.9332082 test1: 0.8980853	best: 0.8980853 (160)	tota
l: 50.8s	remaining: 4m 24s
170:	test: 0.9338582 test1: 0.8993056	best: 0.8993056 (170)	tota
l: 54s	remaining: 4m 21s
180:	test: 0.9341128 test1: 0.8996948	best: 0.8997009 (179)	tota
l: 57.2s	remaining: 4m 19s
190:	test: 0.9351333 test1: 0.9009764	best: 0.9009764 (190)	tota
l: 1m	remaining: 4m 17s
200:	test: 0.9362062 test1: 0.9025007	best: 0.9025007 (200)	tota
l: 1m 4s	remaining: 4m 14s
210:	test: 0.9369944 test1: 0.9039879	best: 0.9039947 (209)	tota
l: 1m 7s	remaining: 4m 11s
220:	test: 0.9378246 test1: 0.9046526	best: 0.9046526 (220)	tota
l: 1m 10s	remaining: 4m 8s
230:	test: 0.9386972 test1: 0.9059646	best: 0.9059646 (230)	tota
```

```
l: 1m 14s       remaining: 4m 6s
240:    test: 0.9391340 test1: 0.9062611        best: 0.9063319 (236)   tota
l: 1m 17s       remaining: 4m 3s
250:    test: 0.9398466 test1: 0.9069987        best: 0.9069987 (250)   tota
l: 1m 20s       remaining: 4m 1s
260:    test: 0.9406335 test1: 0.9080089        best: 0.9080089 (260)   tota
l: 1m 24s       remaining: 3m 59s
270:    test: 0.9412568 test1: 0.9083786        best: 0.9083786 (270)   tota
l: 1m 27s       remaining: 3m 56s
280:    test: 0.9416828 test1: 0.9088382        best: 0.9088382 (280)   tota
l: 1m 31s       remaining: 3m 53s
290:    test: 0.9422108 test1: 0.9094944        best: 0.9094947 (289)   tota
l: 1m 34s       remaining: 3m 50s
300:    test: 0.9432544 test1: 0.9102447        best: 0.9102447 (300)   tota
l: 1m 38s       remaining: 3m 47s
310:    test: 0.9438802 test1: 0.9112836        best: 0.9112836 (310)   tota
l: 1m 41s       remaining: 3m 44s
320:    test: 0.9444680 test1: 0.9119065        best: 0.9119065 (320)   tota
l: 1m 45s       remaining: 3m 42s
330:    test: 0.9452172 test1: 0.9127402        best: 0.9127993 (328)   tota
l: 1m 48s       remaining: 3m 39s
340:    test: 0.9459752 test1: 0.9134512        best: 0.9134512 (340)   tota
l: 1m 51s       remaining: 3m 36s
350:    test: 0.9467804 test1: 0.9143991        best: 0.9143991 (350)   tota
l: 1m 55s       remaining: 3m 33s
360:    test: 0.9468959 test1: 0.9145870        best: 0.9145870 (360)   tota
l: 1m 58s       remaining: 3m 29s
370:    test: 0.9471369 test1: 0.9149109        best: 0.9150023 (368)   tota
l: 2m 1s        remaining: 3m 26s
380:    test: 0.9481843 test1: 0.9156507        best: 0.9156777 (379)   tota
l: 2m 5s        remaining: 3m 23s
390:    test: 0.9482700 test1: 0.9158624        best: 0.9158624 (390)   tota
l: 2m 8s        remaining: 3m 20s
400:    test: 0.9483924 test1: 0.9161037        best: 0.9161037 (400)   tota
l: 2m 11s       remaining: 3m 16s
410:    test: 0.9485282 test1: 0.9162203        best: 0.9162325 (406)   tota
l: 2m 14s       remaining: 3m 13s
420:    test: 0.9489996 test1: 0.9166062        best: 0.9166062 (420)   tota
l: 2m 18s       remaining: 3m 10s
430:    test: 0.9493751 test1: 0.9170448        best: 0.9170448 (430)   tota
l: 2m 21s       remaining: 3m 6s
440:    test: 0.9499920 test1: 0.9175352        best: 0.9175352 (440)   tota
l: 2m 24s       remaining: 3m 3s
450:    test: 0.9502063 test1: 0.9179274        best: 0.9179592 (445)   tota
l: 2m 27s       remaining: 2m 59s
460:    test: 0.9502202 test1: 0.9179548        best: 0.9179592 (445)   tota
l: 2m 30s       remaining: 2m 56s
470:    test: 0.9502453 test1: 0.9179917        best: 0.9179917 (470)   tota
l: 2m 33s       remaining: 2m 52s
480:    test: 0.9502630 test1: 0.9180119        best: 0.9180119 (479)   tota
l: 2m 37s       remaining: 2m 49s
490:    test: 0.9502759 test1: 0.9180311        best: 0.9180311 (489)   tota
l: 2m 40s       remaining: 2m 46s
500:    test: 0.9502838 test1: 0.9180420        best: 0.9180423 (496)   tota
l: 2m 43s       remaining: 2m 42s
510:    test: 0.9502918 test1: 0.9180495        best: 0.9180510 (506)   tota
l: 2m 46s       remaining: 2m 38s
520:    test: 0.9503058 test1: 0.9180313        best: 0.9180510 (506)   tota
l: 2m 49s       remaining: 2m 35s
530:    test: 0.9503094 test1: 0.9180329        best: 0.9180510 (506)   tota
l: 2m 52s       remaining: 2m 32s
```

```
540:	test: 0.9503048 test1: 0.9180237	best: 0.9180510 (506)	tota
l: 2m 55s	remaining: 2m 28s
550:	test: 0.9503038 test1: 0.9180205	best: 0.9180510 (506)	tota
l: 2m 58s	remaining: 2m 25s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.9180509792
bestIteration = 506

Shrink model to first 507 iterations.
```

Out[68]:

```
<catboost.core.CatBoostClassifier at 0xf85444f550>
```

Задание 0:

- bestTest = 0.8827161236
- bestIteration = 419

Задание 3:

- bestTest = 0.9180509792
- bestIteration = 506

Вывод:

- Добавление новых признаков (Задание 3) значительно улучшило качество модели по сравнению с базовым решением.

Задание 4:

Создать признаки на основе отношения: TransactionAmt к вычисленной статистике. Статистика - среднее значение / стандартное отклонение TransactionAmt, сгруппированное по card1 - card6, addr1, addr2, и по признакам, созданным в задании 2.

В [69]:

```
# Leveraging Machine Learning to Detect Fraud: Tips to Developing a Winning Kaggle Solution
# https://developer.nvidia.com/blog/leveraging-machine-learning-to-detect-fraud-tips-to-dev
# temp = df.groupby('card1')['TransactionAmt'].agg(['mean']).rename({'mean':'TransactionAmt
# df = pd.merge(df,temp,on='card1',how='left')
```

В [70]:

```
x_train_task_4 = []
x_test_task_4 = []
x_train_task_4 = x_train_task_3.copy()
x_test_task_4 = x_test_task_3.copy()
```

В [71]:

```
temp = x_train_task_4.groupby('card1')['TransactionAmt'].agg(['mean']).rename({'mean':'Tran
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card1',how='left')
temp = x_train_task_4.groupby('card2')['TransactionAmt'].agg(['mean']).rename({'mean':'Tran
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card2',how='left')
temp = x_train_task_4.groupby('card3')['TransactionAmt'].agg(['mean']).rename({'mean':'Tran
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card3',how='left')
temp = x_train_task_4.groupby('card5')['TransactionAmt'].agg(['mean']).rename({'mean':'Tran
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card5',how='left')
temp = x_train_task_4.groupby('card4')['TransactionAmt'].agg(['mean']).rename({'mean':'Tran
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card4',how='left')
temp = x_train_task_4.groupby('card6')['TransactionAmt'].agg(['mean']).rename({'mean':'Tran
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card6',how='left')
```

В [72]:

```
temp = x_train_task_4.groupby('card1_card2')['TransactionAmt'].agg(['mean']).\
rename({'mean':'TransactionAmt_card1_card2_mean'},axis=1)
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card1_card2',how='left')

temp = x_train_task_4.groupby('card1_card2_card_3_card_5')['TransactionAmt'].agg(['mean']).
rename({'mean':'TransactionAmt_card1_card2_card_3_card_5_mean'},axis=1)
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card1_card2_card_3_card_5',how='left')

temp = x_train_task_4.groupby('card1_card2_card_3_card_5_addr1_addr2')['TransactionAmt'].ag
rename({'mean':'TransactionAmt_card1_card2_card_3_card_5_addr1_addr2_mean'},axis=1)
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card1_card2_card_3_card_5_addr1_addr2',ho
```

В [73]:

```
x_train_task_4.head(2)
```

Out[73]:

	TransactionDT	TransactionAmt	card1	card2	card3	card5	addr1	addr2	C1	C2	C3	C4
0	2916619	218.0	6892	560.0	150.0	226.0	433.0	87.0	3.0	2.0	0.0	0.0
1	2600138	50.0	2922	583.0	150.0	226.0	299.0	87.0	1.0	1.0	0.0	1.0

В [74]:

```
temp = x_test_task_4.groupby('card1')['TransactionAmt'].agg(['mean']).rename({'mean':'Trans
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card1',how='left')
temp = x_test_task_4.groupby('card2')['TransactionAmt'].agg(['mean']).rename({'mean':'Trans
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card2',how='left')
temp = x_train_task_4.groupby('card3')['TransactionAmt'].agg(['mean']).rename({'mean':'Tran
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card3',how='left')
temp = x_test_task_4.groupby('card5')['TransactionAmt'].agg(['mean']).rename({'mean':'Trans
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card5',how='left')
temp = x_test_task_4.groupby('card4')['TransactionAmt'].agg(['mean']).rename({'mean':'Trans
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card4',how='left')
temp = x_test_task_4.groupby('card6')['TransactionAmt'].agg(['mean']).rename({'mean':'Trans
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card6',how='left')
```

В [75]:

```
temp = x_test_task_4.groupby('card1_card2')['TransactionAmt'].agg(['mean']).\
rename({'mean':'TransactionAmt_card1_card2_mean'},axis=1)
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card1_card2',how='left')

temp = x_test_task_4.groupby('card1_card2_card_3_card_5')['TransactionAmt'].agg(['mean']).\
rename({'mean':'TransactionAmt_card1_card2_card_3_card_5_mean'},axis=1)
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card1_card2_card_3_card_5',how='left')

temp = x_test_task_4.groupby('card1_card2_card_3_card_5_addr1_addr2')['TransactionAmt'].agg
rename({'mean':'TransactionAmt_card1_card2_card_3_card_5_addr1_addr2_mean'},axis=1)
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card1_card2_card_3_card_5_addr1_addr2',how=
```

В [76]:

```
x_test_task_4.head(2)
```

Out[76]:

	TransactionDT	TransactionAmt	card1	card2	card3	card5	addr1	addr2	C1	C2	C3	C4
0	1712256	171.0	15186	480.0	150.0	224.0	299.0	87.0	1.0	1.0	0.0	0.0
1	108545	50.0	6019	583.0	150.0	226.0	225.0	87.0	1.0	1.0	0.0	0.0

В [77]:

```
categorical_features = [#'card1_card2',
#   'card1_card2_card_3_card_5',
#   'card1_card2_card_3_card_5_addr1_addr2',
#   'card1_freq_enc',
#   'card2_freq_enc',
#   'card3_freq_enc',
#   'card4_freq_enc',
#   'card5_freq_enc',
#   'card6_freq_enc',
#   'addr1_freq_enc',
#   'addr2_freq_enc',
#   'card4',
#   'card6',
 'TransactionAmt_card1_mean',
 'TransactionAmt_card2_mean',
 'TransactionAmt_card3_mean',
 'TransactionAmt_card5_mean',
 'TransactionAmt_card4_mean',
 'TransactionAmt_card6_mean',
 'TransactionAmt_card1_card2_mean',
 'TransactionAmt_card1_card2_card_3_card_5_mean',
 'TransactionAmt_card1_card2_card_3_card_5_addr1_addr2_mean',
]
```

В [78]:

```
x_train_task_4[categorical_features] = x_train_task_4[categorical_features].astype(str)
x_test_task_4[categorical_features] = x_test_task_4[categorical_features].astype(str)
```

CatBoost с категориальными признаками

В [79]:

```
# eval_sets= [
#     (x_train_task_4[xgb_numerical_features + task_1_fields + categorical_features], y_tra
#     (x_test_task_4[xgb_numerical_features + task_1_fields + categorical_features], y_test
# ]
```

В [80]:

```
# cb_model.fit(
#     x_train_task_4[xgb_numerical_features + task_1_fields + categorical_features],
#     y_train,
#     cat_features = categorical_features,
#     eval_set=eval_sets)
```

В [81]:

```
eval_sets= [
    (x_train_task_4[xgb_numerical_features + categorical_features], y_train),
    (x_test_task_4[xgb_numerical_features + categorical_features], y_test)
]
```

B [82]:

```
cb_model.fit(
    x_train_task_4[xgb_numerical_features + categorical_features],
    y_train,
    cat_features = categorical_features,
    eval_set=eval_sets)
```

```
0:	test: 0.6132339 test1: 0.5993341	best: 0.5993341 (0)	to
tal: 668ms	remaining: 11m 7s
10:	test: 0.7899325 test1: 0.7751216	best: 0.7751216 (10)	to
tal: 4.63s	remaining: 6m 55s
20:	test: 0.8026063 test1: 0.7902130	best: 0.7902130 (20)	to
tal: 7.43s	remaining: 5m 46s
30:	test: 0.8366435 test1: 0.8254546	best: 0.8254546 (30)	to
tal: 10.1s	remaining: 5m 14s
40:	test: 0.8593399 test1: 0.8365513	best: 0.8372979 (37)	to
tal: 12.6s	remaining: 4m 55s
50:	test: 0.9035088 test1: 0.8426747	best: 0.8427371 (49)	to
tal: 15s	remaining: 4m 39s
60:	test: 0.9083215 test1: 0.8428761	best: 0.8432947 (57)	to
tal: 17.7s	remaining: 4m 33s
70:	test: 0.9104771 test1: 0.8431046	best: 0.8436128 (66)	to
tal: 20.3s	remaining: 4m 25s
80:	test: 0.9122926 test1: 0.8447893	best: 0.8447893 (80)	to
tal: 22.5s	remaining: 4m 15s
90:	test: 0.9162515 test1: 0.8506776	best: 0.8507435 (89)	to
tal: 24.6s	remaining: 4m 6s
100:	test: 0.9180849 test1: 0.8519368	best: 0.8520105 (98)	to
tal: 27.4s	remaining: 4m 3s
110:	test: 0.9210122 test1: 0.8529185	best: 0.8529185 (110)	to
tal: 30.8s	remaining: 4m 6s
120:	test: 0.9227673 test1: 0.8537933	best: 0.8538827 (114)	to
tal: 33.2s	remaining: 4m 1s
130:	test: 0.9239293 test1: 0.8560893	best: 0.8560893 (130)	to
tal: 37.2s	remaining: 4m 6s
140:	test: 0.9252659 test1: 0.8568620	best: 0.8568701 (139)	to
tal: 40.8s	remaining: 4m 8s
150:	test: 0.9261173 test1: 0.8578675	best: 0.8578675 (150)	to
tal: 43.7s	remaining: 4m 5s
160:	test: 0.9276018 test1: 0.8592250	best: 0.8592250 (160)	to
tal: 46.3s	remaining: 4m 1s
170:	test: 0.9285928 test1: 0.8604728	best: 0.8604728 (170)	to
tal: 49s	remaining: 3m 57s
180:	test: 0.9293586 test1: 0.8618337	best: 0.8618337 (180)	to
tal: 52.3s	remaining: 3m 56s
190:	test: 0.9302861 test1: 0.8634769	best: 0.8634769 (190)	to
tal: 55.4s	remaining: 3m 54s
200:	test: 0.9310470 test1: 0.8645977	best: 0.8645977 (200)	to
tal: 59.4s	remaining: 3m 56s
210:	test: 0.9318955 test1: 0.8653178	best: 0.8653178 (210)	to
tal: 1m 4s	remaining: 4m 1s
220:	test: 0.9325970 test1: 0.8663283	best: 0.8663283 (220)	to
tal: 1m 9s	remaining: 4m 3s
230:	test: 0.9333743 test1: 0.8671894	best: 0.8671894 (230)	to
tal: 1m 12s	remaining: 4m 1s
240:	test: 0.9340992 test1: 0.8676968	best: 0.8679869 (234)	to
tal: 1m 17s	remaining: 4m 5s
250:	test: 0.9344228 test1: 0.8683418	best: 0.8683418 (250)	to
tal: 1m 22s	remaining: 4m 6s
260:	test: 0.9351400 test1: 0.8690233	best: 0.8690489 (259)	to
```

```
tal: 1m 26s     remaining: 4m 5s
270:    test: 0.9357344 test1: 0.8694189        best: 0.8694189 (269)   to
tal: 1m 30s     remaining: 4m 3s
280:    test: 0.9362147 test1: 0.8705405        best: 0.8705405 (280)   to
tal: 1m 34s     remaining: 4m 2s
290:    test: 0.9370762 test1: 0.8711100        best: 0.8711100 (290)   to
tal: 1m 39s     remaining: 4m 3s
300:    test: 0.9374963 test1: 0.8714719        best: 0.8714719 (300)   to
tal: 1m 43s     remaining: 4m
310:    test: 0.9380398 test1: 0.8717852        best: 0.8717852 (310)   to
tal: 1m 47s     remaining: 3m 58s
320:    test: 0.9389209 test1: 0.8716804        best: 0.8718424 (316)   to
tal: 1m 52s     remaining: 3m 58s
330:    test: 0.9394054 test1: 0.8719726        best: 0.8720202 (329)   to
tal: 1m 57s     remaining: 3m 57s
340:    test: 0.9396884 test1: 0.8721779        best: 0.8721782 (339)   to
tal: 2m remaining: 3m 53s
350:    test: 0.9407110 test1: 0.8720906        best: 0.8722480 (343)   to
tal: 2m 4s      remaining: 3m 49s
360:    test: 0.9417057 test1: 0.8723223        best: 0.8723223 (360)   to
tal: 2m 7s      remaining: 3m 46s
370:    test: 0.9421241 test1: 0.8725176        best: 0.8725221 (369)   to
tal: 2m 11s     remaining: 3m 42s
380:    test: 0.9423365 test1: 0.8727613        best: 0.8728166 (377)   to
tal: 2m 14s     remaining: 3m 38s
390:    test: 0.9433248 test1: 0.8725851        best: 0.8728269 (382)   to
tal: 2m 19s     remaining: 3m 36s
400:    test: 0.9436933 test1: 0.8723487        best: 0.8728269 (382)   to
tal: 2m 21s     remaining: 3m 31s
410:    test: 0.9446469 test1: 0.8722664        best: 0.8728269 (382)   to
tal: 2m 24s     remaining: 3m 26s
420:    test: 0.9452176 test1: 0.8722559        best: 0.8728269 (382)   to
tal: 2m 26s     remaining: 3m 22s
430:    test: 0.9454239 test1: 0.8724544        best: 0.8728269 (382)   to
tal: 2m 30s     remaining: 3m 19s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8728269204
bestIteration = 382

Shrink model to first 383 iterations.
```

Out[82]:

```
<catboost.core.CatBoostClassifier at 0xf85444f550>
```

Задание 0:

- bestTest = 0.8827161236
- bestIteration = 419

Задание 4:

- bestTest = 0.8728269204
- bestIteration = 382

Вывод:

- Добавление новых признаков (Задание 4) не дало улучшения качества модели по сравнению с базовым решением.

Задание 5:

Создать признаки на основе отношения: D15 к вычисленной статистике. Статистика - среднее значение / стандартное отклонение D15, сгруппированное по card1 - card6, addr1, addr2, и по признакам, созданным в задании 2.

В [83]:

```
x_train_task_5 = []
x_test_task_5 = []
x_train_task_5 = x_train_task_3.copy()
x_test_task_5 = x_test_task_3.copy()
```

В [84]:

```
temp = x_train_task_5.groupby('card1')['D15'].agg(['mean']).rename({'mean':'D15_card1_mean'
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card1',how='left')
temp = x_train_task_5.groupby('card2')['D15'].agg(['mean']).rename({'mean':'D15_card2_mean'
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card2',how='left')
temp = x_train_task_5.groupby('card3')['D15'].agg(['mean']).rename({'mean':'D15_card3_mean'
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card3',how='left')
temp = x_train_task_5.groupby('card5')['D15'].agg(['mean']).rename({'mean':'D15_card5_mean'
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card5',how='left')
temp = x_train_task_5.groupby('card4')['D15'].agg(['mean']).rename({'mean':'D15_card4_mean'
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card4',how='left')
temp = x_train_task_5.groupby('card6')['D15'].agg(['mean']).rename({'mean':'D15_card6_mean'
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card6',how='left')
```

В [85]:

```
temp = x_train_task_5.groupby('card1_card2')['D15'].agg(['mean']).\
rename({'mean':'D15_card1_card2_mean'},axis=1)
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card1_card2',how='left')

temp = x_train_task_5.groupby('card1_card2_card_3_card_5')['D15'].agg(['mean']).\
rename({'mean':'D15_card1_card2_card_3_card_5_mean'},axis=1)
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card1_card2_card_3_card_5',how='left')

temp = x_train_task_5.groupby('card1_card2_card_3_card_5_addr1_addr2')['D15'].agg(['mean'])
rename({'mean':'D15_card1_card2_card_3_card_5_addr1_addr2_mean'},axis=1)
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card1_card2_card_3_card_5_addr1_addr2',ho
```

В [86]:

```
temp = x_test_task_5.groupby('card1')['D15'].agg(['mean']).rename({'mean':'D15_card1_mean'}
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card1',how='left')
temp = x_test_task_5.groupby('card2')['D15'].agg(['mean']).rename({'mean':'D15_card2_mean'}
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card2',how='left')
temp = x_test_task_5.groupby('card3')['D15'].agg(['mean']).rename({'mean':'D15_card3_mean'}
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card3',how='left')
temp = x_test_task_5.groupby('card5')['D15'].agg(['mean']).rename({'mean':'D15_card5_mean'}
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card5',how='left')
temp = x_test_task_5.groupby('card4')['D15'].agg(['mean']).rename({'mean':'D15_card4_mean'}
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card4',how='left')
temp = x_test_task_5.groupby('card6')['D15'].agg(['mean']).rename({'mean':'D15_card6_mean'}
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card6',how='left')
```

В [87]:

```
temp = x_test_task_5.groupby('card1_card2')['D15'].agg(['mean']).\
rename({'mean':'D15_card1_card2_mean'},axis=1)
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card1_card2',how='left')

temp = x_test_task_5.groupby('card1_card2_card_3_card_5')['D15'].agg(['mean']).\
rename({'mean':'D15_card1_card2_card_3_card_5_mean'},axis=1)
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card1_card2_card_3_card_5',how='left')

temp = x_test_task_5.groupby('card1_card2_card_3_card_5_addr1_addr2')['D15'].agg(['mean']).
rename({'mean':'D15_card1_card2_card_3_card_5_addr1_addr2_mean'},axis=1)
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card1_card2_card_3_card_5_addr1_addr2',how=
```

В [88]:

```
categorical_features = [#'card1_card2',
#  'card1_card2_card_3_card_5',
#  'card1_card2_card_3_card_5_addr1_addr2',
#  'card1_freq_enc',
#  'card2_freq_enc',
#  'card3_freq_enc',
#  'card4_freq_enc',
#  'card5_freq_enc',
#  'card6_freq_enc',
#  'addr1_freq_enc',
#  'addr2_freq_enc',
#  'card4',
#  'card6',
 'D15_card1_mean',
 'D15_card2_mean',
 'D15_card3_mean',
 'D15_card5_mean',
 'D15_card4_mean',
 'D15_card6_mean',
 'D15_card1_card2_mean',
 'D15_card1_card2_card_3_card_5_mean',
 'D15_card1_card2_card_3_card_5_addr1_addr2_mean',
]
```

В [89]:

```
x_train_task_5[categorical_features] = x_train_task_5[categorical_features].astype(str)
x_test_task_5[categorical_features] = x_test_task_5[categorical_features].astype(str)
```

В [90]:

```
eval_sets= [
    (x_train_task_5[xgb_numerical_features + categorical_features], y_train),
    (x_test_task_5[xgb_numerical_features + categorical_features], y_test)
]
```

B [91]:

```
cb_model.fit(
    x_train_task_5[xgb_numerical_features + categorical_features],
    y_train,
    cat_features = categorical_features,
    eval_set=eval_sets)
```

```
0:	test: 0.6132339 test1: 0.5993341	best: 0.5993341 (0)	tota
l: 750ms	remaining: 12m 29s
10:	test: 0.7899445 test1: 0.7763569	best: 0.7763569 (10)	tota
l: 5.09s	remaining: 7m 37s
20:	test: 0.8025927 test1: 0.7907131	best: 0.7907131 (20)	tota
l: 8.28s	remaining: 6m 26s
30:	test: 0.8366352 test1: 0.8264842	best: 0.8264842 (30)	tota
l: 10.8s	remaining: 5m 36s
40:	test: 0.8858326 test1: 0.8382467	best: 0.8389378 (38)	tota
l: 13.8s	remaining: 5m 21s
50:	test: 0.8929391 test1: 0.8407348	best: 0.8407348 (50)	tota
l: 17.4s	remaining: 5m 23s
60:	test: 0.8973971 test1: 0.8438248	best: 0.8438248 (60)	tota
l: 20.7s	remaining: 5m 18s
70:	test: 0.9006101 test1: 0.8433957	best: 0.8438248 (60)	tota
l: 23.1s	remaining: 5m 1s
80:	test: 0.9068822 test1: 0.8447442	best: 0.8447442 (80)	tota
l: 27s	remaining: 5m 5s
90:	test: 0.9103680 test1: 0.8465932	best: 0.8466988 (86)	tota
l: 31.5s	remaining: 5m 14s
100:	test: 0.9131515 test1: 0.8493702	best: 0.8494212 (99)	tota
l: 36.1s	remaining: 5m 21s
110:	test: 0.9150835 test1: 0.8513091	best: 0.8513091 (110)	tota
l: 40.3s	remaining: 5m 22s
120:	test: 0.9162404 test1: 0.8527600	best: 0.8527860 (119)	tota
l: 44.1s	remaining: 5m 20s
130:	test: 0.9167563 test1: 0.8528959	best: 0.8529195 (129)	tota
l: 47.8s	remaining: 5m 17s
140:	test: 0.9174175 test1: 0.8534306	best: 0.8534306 (140)	tota
l: 51.5s	remaining: 5m 13s
150:	test: 0.9180271 test1: 0.8541916	best: 0.8541916 (150)	tota
l: 54.5s	remaining: 5m 6s
160:	test: 0.9191474 test1: 0.8561788	best: 0.8561788 (160)	tota
l: 59.1s	remaining: 5m 8s
170:	test: 0.9204345 test1: 0.8573325	best: 0.8573325 (170)	tota
l: 1m 2s	remaining: 5m 1s
180:	test: 0.9214004 test1: 0.8587544	best: 0.8587544 (180)	tota
l: 1m 5s	remaining: 4m 54s
190:	test: 0.9225175 test1: 0.8604371	best: 0.8604371 (190)	tota
l: 1m 8s	remaining: 4m 48s
200:	test: 0.9228196 test1: 0.8609848	best: 0.8610809 (197)	tota
l: 1m 11s	remaining: 4m 43s
210:	test: 0.9241342 test1: 0.8621248	best: 0.8621248 (210)	tota
l: 1m 13s	remaining: 4m 36s
220:	test: 0.9249249 test1: 0.8625125	best: 0.8626045 (219)	tota
l: 1m 17s	remaining: 4m 31s
230:	test: 0.9262495 test1: 0.8634257	best: 0.8634257 (230)	tota
l: 1m 20s	remaining: 4m 29s
240:	test: 0.9266662 test1: 0.8637833	best: 0.8637834 (239)	tota
l: 1m 24s	remaining: 4m 26s
250:	test: 0.9270055 test1: 0.8640823	best: 0.8640823 (250)	tota
l: 1m 28s	remaining: 4m 25s
260:	test: 0.9275469 test1: 0.8645105	best: 0.8645105 (260)	tota
```

```
l: 1m 32s       remaining: 4m 21s
270:    test: 0.9285691 test1: 0.8646943        best: 0.8646943 (270)   tota
l: 1m 36s       remaining: 4m 19s
280:    test: 0.9291202 test1: 0.8650850        best: 0.8651137 (279)   tota
l: 1m 39s       remaining: 4m 13s
290:    test: 0.9296783 test1: 0.8654381        best: 0.8654381 (290)   tota
l: 1m 42s       remaining: 4m 8s
300:    test: 0.9301046 test1: 0.8658816        best: 0.8658816 (300)   tota
l: 1m 45s       remaining: 4m 3s
310:    test: 0.9307143 test1: 0.8663227        best: 0.8663227 (310)   tota
l: 1m 49s       remaining: 4m 1s
320:    test: 0.9317772 test1: 0.8665633        best: 0.8665633 (320)   tota
l: 1m 53s       remaining: 4m
330:    test: 0.9321777 test1: 0.8667112        best: 0.8667894 (327)   tota
l: 1m 57s       remaining: 3m 57s
340:    test: 0.9324708 test1: 0.8671473        best: 0.8671473 (340)   tota
l: 2m 1s        remaining: 3m 54s
350:    test: 0.9329712 test1: 0.8674885        best: 0.8674885 (350)   tota
l: 2m 4s        remaining: 3m 50s
360:    test: 0.9332931 test1: 0.8676214        best: 0.8676356 (355)   tota
l: 2m 9s        remaining: 3m 49s
370:    test: 0.9343530 test1: 0.8677291        best: 0.8678244 (367)   tota
l: 2m 13s       remaining: 3m 45s
380:    test: 0.9345929 test1: 0.8677457        best: 0.8678244 (367)   tota
l: 2m 15s       remaining: 3m 40s
390:    test: 0.9348872 test1: 0.8681846        best: 0.8681846 (390)   tota
l: 2m 20s       remaining: 3m 38s
400:    test: 0.9354377 test1: 0.8685663        best: 0.8685663 (400)   tota
l: 2m 23s       remaining: 3m 34s
410:    test: 0.9363309 test1: 0.8690652        best: 0.8690733 (409)   tota
l: 2m 28s       remaining: 3m 32s
420:    test: 0.9371003 test1: 0.8690680        best: 0.8693551 (416)   tota
l: 2m 31s       remaining: 3m 27s
430:    test: 0.9371834 test1: 0.8692875        best: 0.8693551 (416)   tota
l: 2m 34s       remaining: 3m 23s
440:    test: 0.9387320 test1: 0.8700040        best: 0.8700040 (440)   tota
l: 2m 38s       remaining: 3m 21s
450:    test: 0.9388921 test1: 0.8700599        best: 0.8700599 (450)   tota
l: 2m 44s       remaining: 3m 19s
460:    test: 0.9392364 test1: 0.8701962        best: 0.8701962 (460)   tota
l: 2m 47s       remaining: 3m 15s
470:    test: 0.9394299 test1: 0.8705798        best: 0.8705798 (470)   tota
l: 2m 50s       remaining: 3m 11s
480:    test: 0.9397545 test1: 0.8708304        best: 0.8708304 (480)   tota
l: 2m 54s       remaining: 3m 8s
490:    test: 0.9398781 test1: 0.8708925        best: 0.8709918 (483)   tota
l: 2m 56s       remaining: 3m 3s
500:    test: 0.9403864 test1: 0.8707740        best: 0.8709918 (483)   tota
l: 2m 59s       remaining: 2m 58s
510:    test: 0.9403826 test1: 0.8707521        best: 0.8709918 (483)   tota
l: 3m 1s        remaining: 2m 54s
520:    test: 0.9403820 test1: 0.8707535        best: 0.8709918 (483)   tota
l: 3m 3s        remaining: 2m 49s
530:    test: 0.9403825 test1: 0.8707537        best: 0.8709918 (483)   tota
l: 3m 6s        remaining: 2m 45s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8709918449
bestIteration = 483

Shrink model to first 484 iterations.
```

Out[91]:

```
<catboost.core.CatBoostClassifier at 0xf85444f550>
```

Задание 0:

- bestTest = 0.8827161236
- bestIteration = 419

Задание 5:

- bestTest = 0.8709918449
- bestIteration = 483

Вывод:

- Качество модели немного ниже чем в задании 2, но выше чем в задании 0 и задании 1

Задание 6:

Выделить дробную часть и целую часть признака TransactionAmt в два отдельных признака. После создать отдельных признак - логарифм от TransactionAmt

В [92]:

```
import math
# print(5.1 - int(5.1))
# x = math.modf(3.456)
# print(x[0])
# print(x[1])
```

В [93]:

```
x_train_task_6 = []
x_test_task_6 = []
x_train_task_6 = x_train_task_3.copy()
x_test_task_6 = x_test_task_3.copy()
```

В [94]:

```
import math
print(math.modf(45.8978))

def function(x):
    x = math.modf(x)
    return x[1], x[0]
```

```
(0.8977999999999966, 45.0)
```

В [95]:

```
# x_train_task_1['new_date'],
x_train_task_6['TransactionAmr_intager'], x_train_task_6['TransactionAmr_fractional'] = \
zip(*x_train_task_6['TransactionAmt'].map(function))

# x_test_task_1['new_date'],
x_test_task_6['TransactionAmr_intager'], x_test_task_6['TransactionAmr_fractional'] = \
zip(*x_test_task_6['TransactionAmt'].map(function))
```

В [96]:

```
# x_train_task_6['TransactionAmr_log'] = zip(*x_train_task_6['TransactionAmt'].map(function
x_train_task_6['TransactionAmr_log'] = np.log(x_train_task_6['TransactionAmt'])
x_test_task_6['TransactionAmr_log'] = np.log(x_test_task_6['TransactionAmt'])
```

В [97]:

```
task6_features = [
 'TransactionAmr_intager',
 'TransactionAmr_fractional',
 'TransactionAmr_log',
]
```

В [98]:

```
x_train_task_6[task6_features].head(2)
```

Out[98]:

	TransactionAmr_intager	TransactionAmr_fractional	TransactionAmr_log
141582	218.0	0.0	5.384495
131503	50.0	0.0	3.912023

В [99]:

```
# eval_sets= [
#     (x_train_task_6[xgb_numerical_features + categorical_features], y_train),
#     (x_test_task_6[xgb_numerical_features + categorical_features], y_test)
# ]
eval_sets= [
    (x_train_task_6[xgb_numerical_features + task6_features], y_train),
    (x_test_task_6[xgb_numerical_features + task6_features], y_test)
]
```

В [100]:

```
# cb_model.fit(
#     x_train_task_6[xgb_numerical_features + task6_features],
#     y_train,
#     cat_features = categorical_features,
#     eval_set=eval_sets)
```

В [101]:

```
eval_sets= [
    (x_train_task_6[xgb_numerical_features + task6_features], y_train),
    (x_test_task_6[xgb_numerical_features + task6_features], y_test)
]
```

В [104]:

```
# cb_model = cb.CatBoostClassifier(**cb_params)
# cb_model.fit(x_train_task_6[xgb_numerical_features + task6_features], y_train, eval_set=e
```

В [103]:

```
cb_model.fit(
    x_train_task_6[xgb_numerical_features + task6_features],
    y_train,
    #cat_features = categorical_features,
    eval_set=eval_sets)
```

```
0:	test: 0.6829308 test1: 0.6767559	best: 0.6767559 (0)	tota
l: 174ms	remaining: 2m 53s
10:	test: 0.7729142 test1: 0.7622869	best: 0.7622965 (9)	tota
l: 1.24s	remaining: 1m 51s
20:	test: 0.8279478 test1: 0.8231270	best: 0.8231270 (20)	tota
l: 2.23s	remaining: 1m 43s
30:	test: 0.8414742 test1: 0.8332268	best: 0.8332268 (30)	tota
l: 3.08s	remaining: 1m 36s
40:	test: 0.8490078 test1: 0.8425562	best: 0.8427865 (39)	tota
l: 3.72s	remaining: 1m 27s
50:	test: 0.8524722 test1: 0.8458381	best: 0.8458381 (50)	tota
l: 4.32s	remaining: 1m 20s
60:	test: 0.8575901 test1: 0.8498734	best: 0.8499933 (57)	tota
l: 4.91s	remaining: 1m 15s
70:	test: 0.8582895 test1: 0.8501271	best: 0.8511148 (64)	tota
l: 5.59s	remaining: 1m 13s
80:	test: 0.8625141 test1: 0.8538566	best: 0.8538566 (80)	tota
l: 6.28s	remaining: 1m 11s
90:	test: 0.8654997 test1: 0.8570493	best: 0.8570493 (90)	tota
l: 6.91s	remaining: 1m 8s
100:	test: 0.8685545 test1: 0.8605576	best: 0.8605576 (100)	tota
l: 7.56s	remaining: 1m 7s
110:	test: 0.8696675 test1: 0.8610154	best: 0.8610279 (106)	tota
l: 8.11s	remaining: 1m 4s
120:	test: 0.8713190 test1: 0.8625735	best: 0.8625735 (120)	tota
l: 8.69s	remaining: 1m 3s
130:	test: 0.8732775 test1: 0.8646900	best: 0.8646900 (130)	tota
l: 9.28s	remaining: 1m 1s
140:	test: 0.8747180 test1: 0.8658132	best: 0.8658132 (140)	tota
l: 9.87s	remaining: 1m
150:	test: 0.8761941 test1: 0.8672323	best: 0.8672323 (150)	tota
l: 10.5s	remaining: 59.1s
160:	test: 0.8778482 test1: 0.8687972	best: 0.8687972 (160)	tota
l: 11.3s	remaining: 59.1s
170:	test: 0.8797394 test1: 0.8709394	best: 0.8709510 (169)	tota
l: 12.3s	remaining: 59.4s
180:	test: 0.8810733 test1: 0.8723657	best: 0.8723657 (180)	tota
l: 13s	remaining: 58.8s
190:	test: 0.8822296 test1: 0.8738214	best: 0.8738214 (190)	tota
l: 13.7s	remaining: 58.2s
200:	test: 0.8831831 test1: 0.8745627	best: 0.8745627 (200)	tota
l: 14.4s	remaining: 57.4s
210:	test: 0.8837338 test1: 0.8748745	best: 0.8749198 (206)	tota
l: 15s	remaining: 56.1s
220:	test: 0.8848292 test1: 0.8756852	best: 0.8756852 (220)	tota
l: 15.6s	remaining: 54.9s
230:	test: 0.8857132 test1: 0.8767333	best: 0.8767333 (230)	tota
l: 16.2s	remaining: 54s
240:	test: 0.8860846 test1: 0.8772052	best: 0.8772052 (240)	tota
l: 17s	remaining: 53.6s
250:	test: 0.8866281 test1: 0.8779573	best: 0.8779573 (250)	tota
l: 17.9s	remaining: 53.5s
260:	test: 0.8873535 test1: 0.8784585	best: 0.8784585 (260)	tota
```

```
l: 19s  remaining: 53.8s
270:    test: 0.8880070 test1: 0.8789357        best: 0.8789396 (269)   tota
l: 19.7s        remaining: 53.1s
280:    test: 0.8885363 test1: 0.8792828        best: 0.8792828 (280)   tota
l: 20.3s        remaining: 51.9s
290:    test: 0.8891513 test1: 0.8796245        best: 0.8796245 (290)   tota
l: 20.9s        remaining: 50.9s
300:    test: 0.8895779 test1: 0.8798824        best: 0.8798824 (300)   tota
l: 21.4s        remaining: 49.7s
310:    test: 0.8899194 test1: 0.8801941        best: 0.8801941 (310)   tota
l: 22s  remaining: 48.7s
320:    test: 0.8902665 test1: 0.8805143        best: 0.8805143 (320)   tota
l: 22.5s        remaining: 47.6s
330:    test: 0.8906451 test1: 0.8808342        best: 0.8808342 (330)   tota
l: 23.1s        remaining: 46.6s
340:    test: 0.8910895 test1: 0.8811072        best: 0.8811072 (340)   tota
l: 23.6s        remaining: 45.7s
350:    test: 0.8913818 test1: 0.8812377        best: 0.8812377 (350)   tota
l: 24.2s        remaining: 44.7s
360:    test: 0.8918047 test1: 0.8815479        best: 0.8815479 (360)   tota
l: 24.7s        remaining: 43.8s
370:    test: 0.8921678 test1: 0.8817415        best: 0.8817415 (370)   tota
l: 25.3s        remaining: 42.9s
380:    test: 0.8926238 test1: 0.8820942        best: 0.8821052 (379)   tota
l: 25.9s        remaining: 42s
390:    test: 0.8928945 test1: 0.8823416        best: 0.8823416 (390)   tota
l: 26.4s        remaining: 41.1s
400:    test: 0.8931708 test1: 0.8825326        best: 0.8825348 (397)   tota
l: 26.9s        remaining: 40.2s
410:    test: 0.8933258 test1: 0.8826740        best: 0.8826762 (409)   tota
l: 27.5s        remaining: 39.4s
420:    test: 0.8935168 test1: 0.8828081        best: 0.8828082 (419)   tota
l: 28s  remaining: 38.5s
430:    test: 0.8935877 test1: 0.8828855        best: 0.8828855 (430)   tota
l: 28.5s        remaining: 37.7s
440:    test: 0.8935944 test1: 0.8828839        best: 0.8828861 (437)   tota
l: 29s  remaining: 36.8s
450:    test: 0.8936109 test1: 0.8828873        best: 0.8828945 (443)   tota
l: 29.6s        remaining: 36s
460:    test: 0.8936144 test1: 0.8828809        best: 0.8828945 (443)   tota
l: 30.1s        remaining: 35.2s
470:    test: 0.8936352 test1: 0.8828844        best: 0.8828945 (443)   tota
l: 30.6s        remaining: 34.4s
480:    test: 0.8936506 test1: 0.8828790        best: 0.8828945 (443)   tota
l: 31.2s        remaining: 33.6s
490:    test: 0.8936570 test1: 0.8828777        best: 0.8828945 (443)   tota
l: 31.7s        remaining: 32.9s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8828945346
bestIteration = 443

Shrink model to first 444 iterations.
```

Out[103]:

```
<catboost.core.CatBoostClassifier at 0xf8cdc4e1f0>
```

Задание 0 (без обработки):

- bestTest = 0.8827161236

- bestIteration = 419

Задание 1:

- bestTest = 0.8812417137
- bestIteration = 455

Вывод:

- Добавление новых признаков (Задание 1) не дало улучшения качества модели по сравнению с базовым решением.

Задание 2:

- bestTest = 0.9216976237
- bestIteration = 557

Вывод:

- Добавление новых признаков (Задание 2) значительно улучшило качество модели по сравнению с базовым решением.

Задание 3:

- bestTest = 0.9180509792
- bestIteration = 506

Вывод:

- Добавление новых признаков (Задание 3) значительно улучшило качество модели по сравнению с базовым решением.

Задание 4:

- bestTest = 0.8728269204
- bestIteration = 382

Вывод:

- Добавление новых признаков (Задание 4) не дало улучшения качества модели по сравнению с базовым решением.

Задание 5:

- bestTest = 0.8709918449
- bestIteration = 483

Вывод:

- Добавление новых признаков (Задание 5) не дало улучшения качества модели по сравнению с базовым решением.

Задание 6:

- bestTest = 0.8828945346
- bestIteration = 443

Вывод:

- Добавление новых признаков (Задание 6) незначительно качества модели по сравнению с базовым решением.

Задание 7 (опция):

Выполнить предварительную подготовку / очистку признаков P_emaildomain и R_emaildomain (что и как делать - остается на ваше усмотрение) и сделать Frequency Encoding для очищенных признаков.

В [107]:

```
x_train_task_7 = []
x_test_task_7 = []
x_train_task_7 = x_train.copy()
x_test_task_7 = x_test.copy()
```

В [108]:

```
x_train_task_7[['P_emaildomain', 'R_emaildomain']]
```

Out[108]:

	P_emaildomain	R_emaildomain
141582	Unknown	Unknown
131503	yahoo.com	yahoo.com
173925	Unknown	Unknown
177012	aol.com	Unknown
69958	Unknown	Unknown
...	...	...
4848	anonymous.com	Unknown
14879	Unknown	anonymous.com
36680	Unknown	Unknown
118456	gmail.com	Unknown
5139	yahoo.com	Unknown

135000 rows × 2 columns

В [ ]:

```
x_test_task_7[['P_emaildomain', 'R_emaildomain']]
```

В [ ]:

```

```