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:

- Относительно большое улучшение модели по сравнению с базовым решением дали созданные признаки из **Задания 2, 3, 7**.
- Улучшение модели по сравнению с базовым решением дали созданные признаки из **Задания 4, 5, 6**.
- Улучшение модели по сравнению с базовым решением не дали созданные признаки из **Задания 1**

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 = 0.8842896115
- bestIteration = 442

Вывод:

- Добавление новых признаков (Задание 4) незначительно улучшило качества модели по сравнению с базовым решением.

Задание 5:

- bestTest = 0.8832494667
- bestIteration = 463

Вывод:

- Добавление новых признаков (Задание 5) незначительно улучшило качества модели по сравнению с базовым решением.

Задание 6:

- bestTest = 0.8828945346
- bestIteration = 443

Вывод:

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

Задание 7:

- bestTest = 0.8859097396
- bestIteration = 458

Вывод:

- Добавление новых признаков (Задание 7) улучшило 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_score

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 анализ данных. Платформа Kaggle II\Урок 5. Feat
ure Engineering, Feature Selection, part I\HW

29.03.2021  12:40    <DIR>          .
29.03.2021  12:40    <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
29.03.2021  12:17           289 490 lesson_5_hw - 2021-03-29_1.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  12:38           289 544 lesson_5_hw.ipynb
29.03.2021  12:38         1 856 738 lesson_5_hw.pdf
29.03.2021  12:40           471 203 lesson_5_hw.rar
               7 файлов      3 428 271 байт
               4 папок  70 679 908 352 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 столбцов
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	card4	card5	card6	addr1	addr2	dist1	dist2	P
0	2987000	0	86400	68.5	W	13926	NaN	150.0	discover	142.0	credit	315.0	87.0	19.0	NaN	
1	2987001	0	86401	29.0	W	2755	404.0	150.0	mastercard	102.0	credit	325.0	87.0	NaN	NaN	

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	card4	card5	card6	addr1	addr2	dist1	dist2	P_ema
0	3287000	1	7415038	226.0	W	12473	555.0	150.0	visa	226.0	credit	299.0	87.0	116.0	NaN	
1	3287001	0	7415054	3072.0	W	15651	417.0	150.0	visa	226.0	debit	330.0	87.0	NaN	NaN	y

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 проварьировать).

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 сопоставляем целое число (номер категории) - https://dyakonov.org/2016/08/03/python-категориальные-пр
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', 'M9_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/2016/08/03/python-категориальные-пр
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_features])[:,1]
test_scores["XGBoost_gbtree_num_features"] = xgb_model.predict_proba(x_test[xgb_numerical_features])[:,1]
```

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)	total: 446ms	remaining: 7m 25s
10:	test: 0.7782015 test1: 0.7634376	best: 0.7687531 (7)	total: 1.49s	remaining: 2m 14s
20:	test: 0.8199294 test1: 0.8049216	best: 0.8049216 (20)	total: 2.71s	remaining: 2m 6s
30:	test: 0.8359524 test1: 0.8252769	best: 0.8252769 (30)	total: 3.58s	remaining: 1m 51s
40:	test: 0.8482519 test1: 0.8384418	best: 0.8384418 (40)	total: 4.54s	remaining: 1m 46s
50:	test: 0.8514080 test1: 0.8403066	best: 0.8403928 (47)	total: 5.4s	remaining: 1m 40s
60:	test: 0.8539646 test1: 0.8411689	best: 0.8411689 (60)	total: 6.05s	remaining: 1m 33s
70:	test: 0.8557345 test1: 0.8428050	best: 0.8431332 (69)	total: 6.75s	remaining: 1m 28s
80:	test: 0.8603778 test1: 0.8481003	best: 0.8481003 (80)	total: 7.44s	remaining: 1m 24s
90:	test: 0.8657723 test1: 0.8548091	best: 0.8548091 (90)	total: 8.05s	remaining: 1m 20s
100:	test: 0.8678208 test1: 0.8568615	best: 0.8568615 (100)	total: 8.58s	remaining: 1m 16s
110:	test: 0.8698583 test1: 0.8591075	best: 0.8591075 (110)	total: 9.16s	remaining: 1m 13s
120:	test: 0.8716381 test1: 0.8608349	best: 0.8608349 (120)	total: 9.72s	remaining: 1m 10s
130:	test: 0.8728403 test1: 0.8624763	best: 0.8624942 (129)	total: 10.3s	remaining: 1m 8s
140:	test: 0.8741322 test1: 0.8640435	best: 0.8640435 (140)	total: 10.9s	remaining: 1m 6s
150:	test: 0.8754839 test1: 0.8646731	best: 0.8646731 (150)	total: 11.5s	remaining: 1m 4s
160:	test: 0.8769138 test1: 0.8658259	best: 0.8658259 (160)	total: 12.1s	remaining: 1m 2s
170:	test: 0.8789194 test1: 0.8682079	best: 0.8682079 (170)	total: 12.7s	remaining: 1m 1s
180:	test: 0.8802028 test1: 0.8697750	best: 0.8697750 (180)	total: 13.3s	remaining: 1m
190:	test: 0.8821344 test1: 0.8718809	best: 0.8718809 (190)	total: 13.9s	remaining: 58.9s
200:	test: 0.8834463 test1: 0.8735594	best: 0.8735594 (200)	total: 14.5s	remaining: 57.8s
210:	test: 0.8846544 test1: 0.8744413	best: 0.8744413 (210)	total: 15.2s	remaining: 56.7s
220:	test: 0.8852716 test1: 0.8753191	best: 0.8753225 (219)	total: 15.7s	remaining: 55.5s
230:	test: 0.8860888 test1: 0.8760634	best: 0.8760634 (230)	total: 16.3s	remaining: 54.2s
240:	test: 0.8867219 test1: 0.8765119	best: 0.8765119 (240)	total: 16.9s	remaining: 53.2s
250:	test: 0.8871234 test1: 0.8769862	best: 0.8769862 (250)	total: 17.5s	remaining: 52.1s
260:	test: 0.8875339 test1: 0.8774721	best: 0.8774721 (260)	total: 18s	remaining: 51.1s
270:	test: 0.8881798 test1: 0.8780760	best: 0.8780760 (270)	total: 18.7s	remaining: 50.3s
280:	test: 0.8886019 test1: 0.8783551	best: 0.8783551 (280)	total: 19.4s	remaining: 49.6s
290:	test: 0.8890933 test1: 0.8787014	best: 0.8787014 (290)	total: 20.2s	remaining: 49.1s
300:	test: 0.8895511 test1: 0.8790383	best: 0.8790383 (300)	total: 21s	remaining: 48.7s
310:	test: 0.8899169 test1: 0.8792089	best: 0.8792089 (310)	total: 21.7s	remaining: 48.1s
320:	test: 0.8902279 test1: 0.8795504	best: 0.8795504 (320)	total: 22.5s	remaining: 47.6s
330:	test: 0.8908921 test1: 0.8803104	best: 0.8803115 (329)	total: 23.3s	remaining: 47s
340:	test: 0.8913042 test1: 0.8807369	best: 0.8807369 (340)	total: 24.3s	remaining: 46.9s
350:	test: 0.8917350 test1: 0.8810404	best: 0.8810404 (350)	total: 25.1s	remaining: 46.4s
360:	test: 0.8919213 test1: 0.8812479	best: 0.8812479 (360)	total: 25.7s	remaining: 45.5s
370:	test: 0.8922800 test1: 0.8814699	best: 0.8814919 (369)	total: 26.2s	remaining: 44.4s
380:	test: 0.8926446 test1: 0.8817711	best: 0.8817773 (378)	total: 26.8s	remaining: 43.5s
390:	test: 0.8930844 test1: 0.8820645	best: 0.8820645 (390)	total: 27.3s	remaining: 42.5s
400:	test: 0.8935259 test1: 0.8824410	best: 0.8824410 (400)	total: 27.8s	remaining: 41.6s
410:	test: 0.8937299 test1: 0.8826171	best: 0.8826171 (410)	total: 28.4s	remaining: 40.7s
420:	test: 0.8938340 test1: 0.8827120	best: 0.8827161 (419)	total: 28.9s	remaining: 39.8s
430:	test: 0.8938414 test1: 0.8827093	best: 0.8827161 (419)	total: 29.4s	remaining: 38.9s
440:	test: 0.8938376 test1: 0.8826936	best: 0.8827161 (419)	total: 29.9s	remaining: 37.9s
450:	test: 0.8938400 test1: 0.8826844	best: 0.8827161 (419)	total: 30.5s	remaining: 37.1s
460:	test: 0.8938429 test1: 0.8826791	best: 0.8827161 (419)	total: 31.2s	remaining: 36.4s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8827161236
bestIteration = 419

Shrink model to first 420 iterations.
```

Out[26]:
```
<catboost.core.CatBoostClassifier at 0x4e46750a90>
```

В [27]:
```
train_scores["CatBoost_num_features"] = cb_model.predict_proba(x_train[xgb_numerical_features])[:,1]
test_scores["CatBoost_num_features"] = cb_model.predict_proba(x_test[xgb_numerical_features])[:,1]
```

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['TransactionDT'].map(functio
# 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].copy()
df_test_new_task_1 = df_test_new[['isFraud'] + xgb_numerical_features + catigorical_features].copy()

# 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['hour'], x_train_task_1['day
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['hour'], x_test_task_1['day']  =
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_test_new_task_1['hour'], 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_sets)
```

```
0:	test: 0.6667114 test1: 0.6618119	best: 0.6618119 (0)	total: 190ms	remaining: 3m 9s
10:	test: 0.7583015 test1: 0.7459275	best: 0.7459275 (10)	total: 1.47s	remaining: 2m 12s
20:	test: 0.8245631 test1: 0.8100912	best: 0.8100912 (20)	total: 2.43s	remaining: 1m 53s
30:	test: 0.8459280 test1: 0.8345029	best: 0.8345029 (30)	total: 3.21s	remaining: 1m 40s
40:	test: 0.8517896 test1: 0.8401610	best: 0.8401610 (40)	total: 3.81s	remaining: 1m 29s
50:	test: 0.8557151 test1: 0.8445617	best: 0.8445617 (50)	total: 4.67s	remaining: 1m 26s
60:	test: 0.8568242 test1: 0.8454821	best: 0.8459765 (57)	total: 5.5s	remaining: 1m 24s
70:	test: 0.8597014 test1: 0.8481595	best: 0.8481595 (70)	total: 6.18s	remaining: 1m 20s
80:	test: 0.8633124 test1: 0.8520402	best: 0.8521644 (79)	total: 6.86s	remaining: 1m 17s
90:	test: 0.8661622 test1: 0.8550718	best: 0.8550718 (90)	total: 7.57s	remaining: 1m 15s
100:	test: 0.8678117 test1: 0.8566115	best: 0.8566115 (100)	total: 8.28s	remaining: 1m 13s
110:	test: 0.8693647 test1: 0.8588993	best: 0.8588993 (110)	total: 8.92s	remaining: 1m 11s
120:	test: 0.8701545 test1: 0.8594978	best: 0.8594978 (120)	total: 9.51s	remaining: 1m 9s
130:	test: 0.8710036 test1: 0.8603732	best: 0.8603732 (130)	total: 10.1s	remaining: 1m 7s
140:	test: 0.8727777 test1: 0.8620637	best: 0.8620637 (140)	total: 10.7s	remaining: 1m 5s
150:	test: 0.8752387 test1: 0.8648728	best: 0.8648728 (150)	total: 12s	remaining: 1m 7s
160:	test: 0.8772954 test1: 0.8666369	best: 0.8666369 (160)	total: 13.4s	remaining: 1m 9s
170:	test: 0.8796656 test1: 0.8691725	best: 0.8691725 (170)	total: 14.5s	remaining: 1m 10s
180:	test: 0.8809885 test1: 0.8705277	best: 0.8705277 (180)	total: 15.2s	remaining: 1m 8s
190:	test: 0.8819119 test1: 0.8711312	best: 0.8711312 (190)	total: 15.9s	remaining: 1m 7s
200:	test: 0.8831271 test1: 0.8720865	best: 0.8720865 (200)	total: 16.7s	remaining: 1m 6s
210:	test: 0.8837480 test1: 0.8727085	best: 0.8727085 (210)	total: 17.2s	remaining: 1m 4s
220:	test: 0.8847731 test1: 0.8738636	best: 0.8738636 (220)	total: 17.9s	remaining: 1m 3s
230:	test: 0.8859376 test1: 0.8749963	best: 0.8750045 (229)	total: 18.5s	remaining: 1m 1s
240:	test: 0.8865317 test1: 0.8753854	best: 0.8754155 (238)	total: 19.3s	remaining: 1m
250:	test: 0.8872459 test1: 0.8758372	best: 0.8758372 (250)	total: 20.3s	remaining: 1m
260:	test: 0.8876787 test1: 0.8761922	best: 0.8761922 (260)	total: 21.1s	remaining: 59.8s
270:	test: 0.8879844 test1: 0.8763156	best: 0.8763220 (269)	total: 21.9s	remaining: 58.9s
280:	test: 0.8887235 test1: 0.8769977	best: 0.8769977 (280)	total: 22.5s	remaining: 57.5s
290:	test: 0.8893078 test1: 0.8773692	best: 0.8773823 (288)	total: 23.3s	remaining: 56.7s
300:	test: 0.8898077 test1: 0.8779091	best: 0.8779091 (300)	total: 23.9s	remaining: 55.6s
310:	test: 0.8903706 test1: 0.8785188	best: 0.8785242 (309)	total: 24.5s	remaining: 54.3s
320:	test: 0.8907192 test1: 0.8787738	best: 0.8787738 (320)	total: 25s	remaining: 53s
330:	test: 0.8912989 test1: 0.8792003	best: 0.8792003 (330)	total: 25.6s	remaining: 51.8s
340:	test: 0.8917896 test1: 0.8795784	best: 0.8795784 (340)	total: 26.2s	remaining: 50.6s
350:	test: 0.8922154 test1: 0.8798731	best: 0.8798731 (350)	total: 26.8s	remaining: 49.5s
360:	test: 0.8924780 test1: 0.8800431	best: 0.8800431 (360)	total: 27.5s	remaining: 48.8s
370:	test: 0.8927055 test1: 0.8802442	best: 0.8802442 (370)	total: 28.4s	remaining: 48.1s
380:	test: 0.8931425 test1: 0.8806171	best: 0.8806171 (380)	total: 29.1s	remaining: 47.3s
390:	test: 0.8934096 test1: 0.8809053	best: 0.8809053 (390)	total: 30s	remaining: 46.7s
400:	test: 0.8936037 test1: 0.8810861	best: 0.8810861 (400)	total: 30.7s	remaining: 45.9s
410:	test: 0.8937058 test1: 0.8811599	best: 0.8811599 (410)	total: 31.3s	remaining: 44.8s
420:	test: 0.8938017 test1: 0.8812334	best: 0.8812334 (420)	total: 31.8s	remaining: 43.7s
430:	test: 0.8938038 test1: 0.8812265	best: 0.8812334 (420)	total: 32.3s	remaining: 42.6s
440:	test: 0.8938145 test1: 0.8812315	best: 0.8812334 (420)	total: 32.8s	remaining: 41.6s
450:	test: 0.8938250 test1: 0.8812374	best: 0.8812377 (449)	total: 33.4s	remaining: 40.6s
460:	test: 0.8938320 test1: 0.8812370	best: 0.8812417 (455)	total: 33.9s	remaining: 39.6s
470:	test: 0.8938257 test1: 0.8812249	best: 0.8812417 (455)	total: 34.4s	remaining: 38.6s
480:	test: 0.8938242 test1: 0.8812157	best: 0.8812417 (455)	total: 34.9s	remaining: 37.7s
490:	test: 0.8938261 test1: 0.8812111	best: 0.8812417 (455)	total: 35.4s	remaining: 36.7s
500:	test: 0.8938288 test1: 0.8812090	best: 0.8812417 (455)	total: 36s	remaining: 35.8s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8812417137
bestIteration = 455

Shrink model to first 456 iterations.
```

Out[38]: `<catboost.core.CatBoostClassifier at 0x4e4ca28bb0>`

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

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_numerical_features + task_1_fields])
test_scores["CatBoost_task1_features"] = cb_model.predict_proba(x_test_task_1[xgb_numerical_features + task_1_fields])[:
#train_scores["CatBoost_num_features"] = cb_model.predict_proba(x_train_task_1[xgb_numerical_features + task_1_fields])[
#test_scores["CatBoost_num_features"] = cb_model.predict_proba(x_test_task_1[xgb_numerical_features + task_1_fields])[:,:
```

В [40]:
```
y_pred = cb_model.predict_proba(df_test_new_task_1[['isFraud'] + xgb_numerical_features + task_1_fields])[:,1]
```

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', 'card5', 'addr1', 'addr2', 'C1', 'C2',
       ...
       'M5', 'M6', 'M7', 'M8', 'M9', 'year', 'month', 'week_day', 'hour', 'day'], dtype='object', length=213)
```

В [44]:
```
# x_train_task_1['card1_card2'] = x_train_task_1.agg(lambda x: x['card1'] + x['card2'], axis=1)
# 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['addr2']}", axis=1)

x_train_task_1['card1_card2'] = x_train_task_1.agg(lambda x: f"{x['card1']} {x['card2']}", axis=1)
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']}", axis=1)

# x_train_task_1[['card1_card2', 'card1_card2_card_3_card_5', 'card1_card2_card_3_card_5_addr1_addr2']].head(2)
```

В [45]:
```
x_train_task_1[['card1_card2', 'card1_card2_card_3_card_5', 'card1_card2_card_3_card_5_addr1_addr2', 'year', 'hour', 'da
```

Out[45]:

	card1_card2	card1_card2_card_3_card_5	card1_card2_card_3_card_5_addr1_addr2	year	hour	day	week_day	month
141582	6892 560.0	6892 560.0 150.0 226.0	6892 560.0 150.0 226.0 433.0 87.0	2017	18	3	4	11
131503	2922 583.0	2922 583.0 150.0 226.0	2922 583.0 150.0 226.0 299.0 87.0	2017	2	31	1	10

В [46]:
```
x_test_task_1['card1_card2'] = x_test_task_1.agg(lambda x: f"{x['card1']} {x['card2']}", axis=1)
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']}", axis=1)
```

В [47]:
```
x_test_task_1[['card1_card2', 'card1_card2_card_3_card_5', 'card1_card2_card_3_card_5_addr1_addr2', 'year', 'hour', 'day
```

Out[47]:

	card1_card2	card1_card2_card_3_card_5	card1_card2_card_3_card_5_addr1_addr2	year	hour	day	week_day	month
78715	15186 480.0	15186 480.0 150.0 224.0	15186 480.0 150.0 224.0 299.0 87.0	2017	19	20	4	10
907	6019 583.0	6019 583.0 150.0 226.0	6019 583.0 150.0 226.0 225.0 87.0	2017	6	2	0	10

В [48]:
```
x_test_task_1['card1_card2'] = x_test_task_1.agg(lambda x: f"{x['card1']} {x['card2']}", axis=1)
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']}", axis=1)
```

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_card_5_addr1_addr2']
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].copy()
# 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_train),
#     (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)	total: 530ms	remaining: 8m 49s
10:	test: 0.7872496 test1: 0.7697118	best: 0.7697118 (10)	total: 3.65s	remaining: 5m 28s
20:	test: 0.8188597 test1: 0.8034291	best: 0.8034291 (20)	total: 5.4s	remaining: 4m 11s
30:	test: 0.8414185 test1: 0.8284577	best: 0.8284577 (30)	total: 9.29s	remaining: 4m 50s
40:	test: 0.8625035 test1: 0.8473969	best: 0.8478863 (39)	total: 12.9s	remaining: 5m
50:	test: 0.9180574 test1: 0.8809312	best: 0.8809312 (50)	total: 15.9s	remaining: 4m 55s
60:	test: 0.9245797 test1: 0.8846792	best: 0.8846792 (60)	total: 18.8s	remaining: 4m 49s
70:	test: 0.9267131 test1: 0.8856667	best: 0.8857395 (68)	total: 21.2s	remaining: 4m 37s
80:	test: 0.9278434 test1: 0.8857564	best: 0.8859937 (76)	total: 23.6s	remaining: 4m 27s
90:	test: 0.9290020 test1: 0.8884479	best: 0.8884479 (90)	total: 26.7s	remaining: 4m 26s
100:	test: 0.9299823 test1: 0.8911254	best: 0.8911254 (100)	total: 29.9s	remaining: 4m 26s
110:	test: 0.9317724 test1: 0.8940895	best: 0.8940895 (110)	total: 33.5s	remaining: 4m 27s
120:	test: 0.9327979 test1: 0.8957085	best: 0.8957085 (120)	total: 36.1s	remaining: 4m 22s
130:	test: 0.9329699 test1: 0.8963070	best: 0.8963070 (130)	total: 38.1s	remaining: 4m 12s
140:	test: 0.9333574 test1: 0.8966002	best: 0.8966983 (139)	total: 40.7s	remaining: 4m 7s
150:	test: 0.9341148 test1: 0.8972911	best: 0.8972911 (150)	total: 43.2s	remaining: 4m 2s
160:	test: 0.9358735 test1: 0.8985740	best: 0.8985740 (160)	total: 46.2s	remaining: 4m
170:	test: 0.9369014 test1: 0.9001801	best: 0.9001801 (170)	total: 48s	remaining: 3m 52s
180:	test: 0.9376979 test1: 0.9016967	best: 0.9016967 (180)	total: 49.9s	remaining: 3m 45s
190:	test: 0.9384334 test1: 0.9030432	best: 0.9030432 (190)	total: 53.1s	remaining: 3m 45s
200:	test: 0.9388313 test1: 0.9037380	best: 0.9037392 (199)	total: 57.1s	remaining: 3m 46s
210:	test: 0.9393869 test1: 0.9049347	best: 0.9049557 (208)	total: 1m 1s	remaining: 3m 50s
220:	test: 0.9399397 test1: 0.9058418	best: 0.9058418 (220)	total: 1m 5s	remaining: 3m 51s
230:	test: 0.9405147 test1: 0.9066829	best: 0.9066829 (230)	total: 1m 9s	remaining: 3m 49s
240:	test: 0.9408963 test1: 0.9073364	best: 0.9073364 (240)	total: 1m 12s	remaining: 3m 49s
250:	test: 0.9415587 test1: 0.9082409	best: 0.9082409 (250)	total: 1m 16s	remaining: 3m 47s
260:	test: 0.9422290 test1: 0.9091341	best: 0.9091341 (260)	total: 1m 19s	remaining: 3m 45s
270:	test: 0.9425660 test1: 0.9095927	best: 0.9095927 (270)	total: 1m 23s	remaining: 3m 44s
280:	test: 0.9430781 test1: 0.9101844	best: 0.9101844 (280)	total: 1m 27s	remaining: 3m 44s
290:	test: 0.9434543 test1: 0.9107689	best: 0.9107689 (290)	total: 1m 32s	remaining: 3m 44s
300:	test: 0.9437471 test1: 0.9112131	best: 0.9112154 (298)	total: 1m 36s	remaining: 3m 43s
310:	test: 0.9441455 test1: 0.9117472	best: 0.9117472 (310)	total: 1m 40s	remaining: 3m 42s
320:	test: 0.9451149 test1: 0.9127553	best: 0.9127553 (320)	total: 1m 42s	remaining: 3m 37s
330:	test: 0.9452018 test1: 0.9130111	best: 0.9130114 (329)	total: 1m 45s	remaining: 3m 32s
340:	test: 0.9457598 test1: 0.9133788	best: 0.9133788 (340)	total: 1m 49s	remaining: 3m 31s
350:	test: 0.9458491 test1: 0.9135031	best: 0.9135031 (350)	total: 1m 53s	remaining: 3m 29s
360:	test: 0.9468890 test1: 0.9144436	best: 0.9144436 (360)	total: 1m 56s	remaining: 3m 26s
370:	test: 0.9471215 test1: 0.9148807	best: 0.9148807 (370)	total: 1m 59s	remaining: 3m 22s
380:	test: 0.9482912 test1: 0.9159234	best: 0.9159234 (380)	total: 2m 2s	remaining: 3m 19s
390:	test: 0.9483800 test1: 0.9161118	best: 0.9161136 (388)	total: 2m 6s	remaining: 3m 17s
400:	test: 0.9497767 test1: 0.9173843	best: 0.9173843 (400)	total: 2m 10s	remaining: 3m 15s
410:	test: 0.9508740 test1: 0.9184315	best: 0.9184315 (410)	total: 2m 13s	remaining: 3m 11s
420:	test: 0.9512871 test1: 0.9188090	best: 0.9188090 (420)	total: 2m 17s	remaining: 3m 8s
430:	test: 0.9515353 test1: 0.9190670	best: 0.9190670 (430)	total: 2m 20s	remaining: 3m 4s
440:	test: 0.9525292 test1: 0.9199369	best: 0.9199369 (440)	total: 2m 23s	remaining: 3m 1s
450:	test: 0.9525703 test1: 0.9199884	best: 0.9199884 (449)	total: 2m 27s	remaining: 2m 59s
460:	test: 0.9526155 test1: 0.9199909	best: 0.9199909 (460)	total: 2m 29s	remaining: 2m 54s
470:	test: 0.9526270 test1: 0.9199953	best: 0.9199972 (467)	total: 2m 32s	remaining: 2m 50s
480:	test: 0.9531521 test1: 0.9205022	best: 0.9205022 (479)	total: 2m 35s	remaining: 2m 47s
490:	test: 0.9531602 test1: 0.9204948	best: 0.9205032 (481)	total: 2m 37s	remaining: 2m 43s
500:	test: 0.9531621 test1: 0.9205005	best: 0.9205033 (497)	total: 2m 38s	remaining: 2m 38s
510:	test: 0.9531664 test1: 0.9204961	best: 0.9205033 (497)	total: 2m 40s	remaining: 2m 34s
520:	test: 0.9531864 test1: 0.9204943	best: 0.9205033 (497)	total: 2m 43s	remaining: 2m 29s
530:	test: 0.9531969 test1: 0.9205009	best: 0.9205033 (497)	total: 2m 45s	remaining: 2m 25s
540:	test: 0.9531981 test1: 0.9204980	best: 0.9205033 (497)	total: 2m 47s	remaining: 2m 22s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.9205033376
bestIteration = 497

Shrink model to first 498 iterations.
```

Out[52]:
```
<catboost.core.CatBoostClassifier at 0x4e4ca28bb0>
```

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 [53]:
```
# train_scores["CatBoost_task2_features"] = \
#     cb_model.predict_proba(x_train_task_1[xgb_numerical_features + task_1_fields + categorical_features])[:,1]
train_scores["CatBoost_task2_features"] = \
    cb_model.predict_proba(x_train_task_1[xgb_numerical_features + categorical_features])[:,1]
```

В [54]:
```
# test_scores["CatBoost_task2_features"] = \
#     cb_model.predict_proba(x_test_task_1[xgb_numerical_features + task_1_fields + categorical_features])[:,1]
test_scores["CatBoost_task2_features"] = \
    cb_model.predict_proba(x_test_task_1[xgb_numerical_features + categorical_features])[:,1]
```

Задание 3:

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

См. "Урок 4 Предварительная обработка признаков/Категориальные признаки/Второй способ". Файл webinar4_features_part1.ipynb.

В [55]:
```
data = []
data_test = []
data = x_train_task_1.copy()
data_test = x_test_task_1.copy()
```

В [56]:
```
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 [57]:
```
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 с результатами.
```

Out[57]:

	card1	card1_freq_enc	card2	card2_freq_enc	card3	card3_freq_enc	card4	card4_freq_enc	card5	card5_freq_enc	card6	card6_freq_enc
141582	6892	0.000311	560.0	0.000436	150.0	0.879139	visa	0.658237	226.0	0.51426	credit	0.317059
131503	2922	0.000104	583.0	0.054646	150.0	0.879139	visa	0.658237	226.0	0.51426	credit	0.317059

В [58]:
```
# 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 [59]:
```
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 с результатами.
```

Out[59]:

	card1	card1_freq_enc	card2	card2_freq_enc	card3	card3_freq_enc	card4	card4_freq_enc	card5	card5_freq_enc	card6	card6_freq_
78715	15186	0.000267	480.0	0.003451	150.0	0.881531	mastercard	0.303644	224.0	0.128824	debit	0.679
907	6019	0.018267	583.0	0.055197	150.0	0.881531	visa	0.654067	226.0	0.515966	credit	0.320

В [118]:
```
# task3_cat_features = ['card1_freq_encode', 'card2_freq_encode', 'card3_freq_encode', \
#      'card4_freq_encode', 'card5_freq_encode', 'card6_freq_encode', 'addr1_freq_encode', 'addr2_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 [61]:
```
x_train_task_3 = data[xgb_numerical_features + task_1_fields + categorical_features].copy()
```

В [62]:
```
x_train_task_3["card4"].head(2)
```

Out[62]:
```
141582    visa
131503    visa
Name: card4, dtype: category
Categories (5, object): ['american express', 'discover', 'mastercard', 'visa', 'Unknown']
```

В [63]:
```
x_train_task_3[categorical_features] = x_train_task_3[categorical_features].astype(str)
```

В [64]:
```
x_test_task_3 = data_test[xgb_numerical_features + task_1_fields + categorical_features].copy()
x_test_task_3[categorical_features] = x_test_task_3[categorical_features].astype(str)
```

В [65]:
```
#x_test_task_3.isnull().sum(axis = 0)
```

В [119]:
```
# eval_sets= [
#     (x_train_task_3[xgb_numerical_features + task_1_fields + categorical_features], y_train),
#     (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 [120]:
```
# 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)     total: 745ms    remaining: 12m 24s
10:     test: 0.7880146 test1: 0.7277898        best: 0.7584509 (8)     total: 4.25s    remaining: 6m 22s
20:     test: 0.8285493 test1: 0.8307867        best: 0.8307867 (20)    total: 7.05s    remaining: 5m 28s
30:     test: 0.8470180 test1: 0.8354330        best: 0.8380251 (28)    total: 10.2s    remaining: 5m 18s
40:     test: 0.8576627 test1: 0.8261190        best: 0.8380251 (28)    total: 13.5s    remaining: 5m 15s
50:     test: 0.8862479 test1: 0.8623361        best: 0.8623361 (50)    total: 16.3s    remaining: 5m 3s
60:     test: 0.9139052 test1: 0.8819011        best: 0.8819011 (60)    total: 19.3s    remaining: 4m 56s
70:     test: 0.9222446 test1: 0.8862534        best: 0.8862534 (70)    total: 22.1s    remaining: 4m 49s
80:     test: 0.9252371 test1: 0.8875602        best: 0.8875602 (80)    total: 25s      remaining: 4m 44s
90:     test: 0.9272170 test1: 0.8892153        best: 0.8893226 (88)    total: 27.9s    remaining: 4m 39s
100:    test: 0.9287967 test1: 0.8920581        best: 0.8920581 (100)   total: 31.1s    remaining: 4m 36s
110:    test: 0.9298694 test1: 0.8937770        best: 0.8937770 (110)   total: 34.1s    remaining: 4m 33s
120:    test: 0.9309668 test1: 0.8952842        best: 0.8952842 (120)   total: 37.3s    remaining: 4m 31s
130:    test: 0.9322053 test1: 0.8971725        best: 0.8971745 (129)   total: 40.7s    remaining: 4m 30s
140:    test: 0.9323883 test1: 0.8973558        best: 0.8973558 (140)   total: 43.7s    remaining: 4m 25s
150:    test: 0.9325430 test1: 0.8977638        best: 0.8977638 (150)   total: 46.8s    remaining: 4m 23s
160:    test: 0.9332082 test1: 0.8980853        best: 0.8980853 (160)   total: 49.9s    remaining: 4m 19s
170:    test: 0.9338582 test1: 0.8993056        best: 0.8993056 (170)   total: 53s      remaining: 4m 16s
180:    test: 0.9341128 test1: 0.8996948        best: 0.8997009 (179)   total: 57.1s    remaining: 4m 18s
190:    test: 0.9351333 test1: 0.9009764        best: 0.9009764 (190)   total: 1m       remaining: 4m 16s
200:    test: 0.9362062 test1: 0.9025007        best: 0.9025007 (200)   total: 1m 4s    remaining: 4m 15s
210:    test: 0.9369944 test1: 0.9039879        best: 0.9039947 (209)   total: 1m 7s    remaining: 4m 12s
220:    test: 0.9378246 test1: 0.9046526        best: 0.9046526 (220)   total: 1m 10s   remaining: 4m 10s
230:    test: 0.9386972 test1: 0.9059646        best: 0.9059646 (230)   total: 1m 14s   remaining: 4m 7s
240:    test: 0.9391340 test1: 0.9062611        best: 0.9063319 (236)   total: 1m 17s   remaining: 4m 4s
250:    test: 0.9398466 test1: 0.9069987        best: 0.9069987 (250)   total: 1m 20s   remaining: 4m
260:    test: 0.9406335 test1: 0.9080089        best: 0.9080089 (260)   total: 1m 24s   remaining: 3m 58s
270:    test: 0.9412568 test1: 0.9083786        best: 0.9083786 (270)   total: 1m 27s   remaining: 3m 54s
280:    test: 0.9416828 test1: 0.9088382        best: 0.9088382 (280)   total: 1m 30s   remaining: 3m 51s
290:    test: 0.9422108 test1: 0.9094944        best: 0.9094947 (289)   total: 1m 33s   remaining: 3m 48s
300:    test: 0.9432544 test1: 0.9102447        best: 0.9102447 (300)   total: 1m 37s   remaining: 3m 45s
310:    test: 0.9438802 test1: 0.9112836        best: 0.9112836 (310)   total: 1m 40s   remaining: 3m 42s
320:    test: 0.9444680 test1: 0.9119065        best: 0.9119065 (320)   total: 1m 43s   remaining: 3m 39s
330:    test: 0.9452172 test1: 0.9127402        best: 0.9127993 (328)   total: 1m 47s   remaining: 3m 36s
340:    test: 0.9459752 test1: 0.9134512        best: 0.9134512 (340)   total: 1m 50s   remaining: 3m 33s
350:    test: 0.9467804 test1: 0.9143991        best: 0.9143991 (350)   total: 1m 53s   remaining: 3m 29s
360:    test: 0.9468959 test1: 0.9145870        best: 0.9145870 (360)   total: 1m 56s   remaining: 3m 26s
370:    test: 0.9471369 test1: 0.9149109        best: 0.9150023 (368)   total: 2m       remaining: 3m 23s
380:    test: 0.9481843 test1: 0.9156507        best: 0.9156777 (379)   total: 2m 3s    remaining: 3m 21s
390:    test: 0.9482700 test1: 0.9158624        best: 0.9158624 (390)   total: 2m 7s    remaining: 3m 17s
400:    test: 0.9483924 test1: 0.9161037        best: 0.9161037 (400)   total: 2m 11s   remaining: 3m 15s
410:    test: 0.9485282 test1: 0.9162203        best: 0.9162325 (406)   total: 2m 14s   remaining: 3m 12s
420:    test: 0.9489996 test1: 0.9166062        best: 0.9166062 (420)   total: 2m 17s   remaining: 3m 9s
430:    test: 0.9493751 test1: 0.9170448        best: 0.9170448 (430)   total: 2m 21s   remaining: 3m 6s
440:    test: 0.9499920 test1: 0.9175352        best: 0.9175352 (440)   total: 2m 24s   remaining: 3m 2s
450:    test: 0.9502063 test1: 0.9179274        best: 0.9179592 (445)   total: 2m 28s   remaining: 3m
460:    test: 0.9502202 test1: 0.9179548        best: 0.9179592 (445)   total: 2m 31s   remaining: 2m 57s
470:    test: 0.9502453 test1: 0.9179917        best: 0.9179917 (470)   total: 2m 34s   remaining: 2m 53s
480:    test: 0.9502630 test1: 0.9180119        best: 0.9180119 (479)   total: 2m 37s   remaining: 2m 50s
490:    test: 0.9502759 test1: 0.9180311        best: 0.9180311 (489)   total: 2m 40s   remaining: 2m 46s
500:    test: 0.9502838 test1: 0.9180420        best: 0.9180423 (496)   total: 2m 43s   remaining: 2m 43s
510:    test: 0.9502918 test1: 0.9180495        best: 0.9180510 (506)   total: 2m 46s   remaining: 2m 39s
520:    test: 0.9503058 test1: 0.9180313        best: 0.9180510 (506)   total: 2m 50s   remaining: 2m 36s
530:    test: 0.9503094 test1: 0.9180329        best: 0.9180510 (506)   total: 2m 53s   remaining: 2m 33s
540:    test: 0.9503048 test1: 0.9180237        best: 0.9180510 (506)   total: 2m 56s   remaining: 2m 29s
550:    test: 0.9503038 test1: 0.9180205        best: 0.9180510 (506)   total: 2m 59s   remaining: 2m 26s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.9180509792
bestIteration = 506

Shrink model to first 507 iterations.
```

Out[120]:
```
<catboost.core.CatBoostClassifier at 0x4e4ca28bb0>
```

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 [68]:
```
# 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-developing-a-winning-kaggle-solu
# temp = df.groupby('card1')['TransactionAmt'].agg(['mean']).rename({'mean':'TransactionAmt_card1_mean'},axis=1)
# df = pd.merge(df,temp,on='card1',how='left')
```

В [121]:
```
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 [122]:
```
temp = x_train_task_4.groupby('card1')['TransactionAmt'].agg(['mean']).rename({'mean':'TransactionAmt_card1_mean'},axis=
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':'TransactionAmt_card2_mean'},axis=
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':'TransactionAmt_card3_mean'},axis=
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':'TransactionAmt_card5_mean'},axis=
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':'TransactionAmt_card4_mean'},axis=
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':'TransactionAmt_card6_mean'},axis=
x_train_task_4 = pd.merge(x_train_task_4,temp,on='card6',how='left')
```

В [123]:
```
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'].agg(['mean']).\
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',how='left')
```

В [124]:
```
x_train_task_4.head(2)
```

Out[124]:

	TransactionDT	TransactionAmt	card1	card2	card3	card5	addr1	addr2	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
0	2916619	218.0	6892	560.0	150.0	226.0	433.0	87.0	3.0	2.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	2.0	0.0	24.0	2.0
1	2600138	50.0	2922	583.0	150.0	226.0	299.0	87.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0

В [125]:
```
temp = x_test_task_4.groupby('card1')['TransactionAmt'].agg(['mean']).rename({'mean':'TransactionAmt_card1_mean'},axis=1
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':'TransactionAmt_card2_mean'},axis=1
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':'TransactionAmt_card3_mean'},axis=
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':'TransactionAmt_card5_mean'},axis=1
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':'TransactionAmt_card4_mean'},axis=1
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':'TransactionAmt_card6_mean'},axis=1
x_test_task_4 = pd.merge(x_test_task_4,temp,on='card6',how='left')
```

В [126]:
```
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(['mean']).\
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='left')
```

В [127]:
```
x_test_task_4.head(2)
```

Out[127]:

	TransactionDT	TransactionAmt	card1	card2	card3	card5	addr1	addr2	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
0	1712256	171.0	15186	480.0	150.0	224.0	299.0	87.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	2.0	0.0	1.0	0.0	14.0	1.0
1	108545	50.0	6019	583.0	150.0	226.0	225.0	87.0	1.0	1.0	0.0	0.0	0.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0

В [128]:
```
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 [129]:
```
###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 [130]:
```
# eval_sets= [
#     (x_train_task_4[xgb_numerical_features + task_1_fields + categorical_features], y_train),
#     (x_test_task_4[xgb_numerical_features + task_1_fields + categorical_features], y_test)
# ]
```

В [131]:
```
# 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 [132]:
```
eval_sets= [
    (x_train_task_4[xgb_numerical_features + categorical_features], y_train),
    (x_test_task_4[xgb_numerical_features + categorical_features], y_test)
]
```

В [133]:
```
cb_model.fit(
    x_train_task_4[xgb_numerical_features + categorical_features],
    y_train,
    ### cat_features = categorical_features,
    eval_set=eval_sets)
```

```
0:	test: 0.6740226 test1: 0.6630859	best: 0.6630859 (0)	total: 205ms	remaining: 3m 24s
10:	test: 0.7893298 test1: 0.7825786	best: 0.7847464 (8)	total: 1.09s	remaining: 1m 37s
20:	test: 0.8253676 test1: 0.8184220	best: 0.8188058 (19)	total: 1.75s	remaining: 1m 21s
30:	test: 0.8499644 test1: 0.8407726	best: 0.8407726 (30)	total: 2.33s	remaining: 1m 12s
40:	test: 0.8547295 test1: 0.8445045	best: 0.8445045 (40)	total: 2.94s	remaining: 1m 8s
50:	test: 0.8558641 test1: 0.8452847	best: 0.8453157 (49)	total: 3.54s	remaining: 1m 5s
60:	test: 0.8595867 test1: 0.8494737	best: 0.8494737 (60)	total: 4.13s	remaining: 1m 3s
70:	test: 0.8622846 test1: 0.8516733	best: 0.8516733 (70)	total: 4.73s	remaining: 1m 1s
80:	test: 0.8643770 test1: 0.8531365	best: 0.8531365 (80)	total: 5.32s	remaining: 1m
90:	test: 0.8663186 test1: 0.8550837	best: 0.8550837 (90)	total: 5.89s	remaining: 58.8s
100:	test: 0.8708580 test1: 0.8598731	best: 0.8598731 (100)	total: 6.48s	remaining: 57.7s
110:	test: 0.8722877 test1: 0.8613814	best: 0.8613814 (110)	total: 7.08s	remaining: 56.7s
120:	test: 0.8736166 test1: 0.8630247	best: 0.8630247 (120)	total: 7.66s	remaining: 55.6s
130:	test: 0.8743437 test1: 0.8633625	best: 0.8633693 (129)	total: 8.24s	remaining: 54.7s
140:	test: 0.8770043 test1: 0.8664144	best: 0.8664144 (140)	total: 8.91s	remaining: 54.3s
150:	test: 0.8787005 test1: 0.8678973	best: 0.8678973 (150)	total: 9.53s	remaining: 53.6s
160:	test: 0.8799900 test1: 0.8693879	best: 0.8693879 (160)	total: 10.1s	remaining: 52.8s
170:	test: 0.8818169 test1: 0.8712703	best: 0.8712703 (170)	total: 10.7s	remaining: 52.1s
180:	test: 0.8832640 test1: 0.8728124	best: 0.8728124 (180)	total: 11.4s	remaining: 51.4s
190:	test: 0.8841112 test1: 0.8734549	best: 0.8734549 (190)	total: 11.9s	remaining: 50.6s
200:	test: 0.8851995 test1: 0.8744137	best: 0.8744137 (200)	total: 12.5s	remaining: 49.8s
210:	test: 0.8863596 test1: 0.8755886	best: 0.8755886 (210)	total: 13.1s	remaining: 49s
220:	test: 0.8875592 test1: 0.8765772	best: 0.8765772 (220)	total: 13.7s	remaining: 48.3s
230:	test: 0.8879670 test1: 0.8768707	best: 0.8768707 (230)	total: 14.3s	remaining: 47.5s
240:	test: 0.8889882 test1: 0.8776054	best: 0.8776054 (240)	total: 14.8s	remaining: 46.7s
250:	test: 0.8897267 test1: 0.8782474	best: 0.8782474 (250)	total: 15.4s	remaining: 46s
260:	test: 0.8903295 test1: 0.8787154	best: 0.8787154 (260)	total: 16s	remaining: 45.2s
270:	test: 0.8911464 test1: 0.8794441	best: 0.8794441 (270)	total: 16.6s	remaining: 44.5s
280:	test: 0.8917696 test1: 0.8799944	best: 0.8799944 (280)	total: 17.1s	remaining: 43.8s
290:	test: 0.8923611 test1: 0.8805816	best: 0.8805816 (290)	total: 17.7s	remaining: 43.1s
300:	test: 0.8930319 test1: 0.8811109	best: 0.8811109 (300)	total: 18.3s	remaining: 42.4s
310:	test: 0.8934333 test1: 0.8814480	best: 0.8814480 (310)	total: 18.8s	remaining: 41.7s
320:	test: 0.8940828 test1: 0.8820215	best: 0.8820215 (320)	total: 19.4s	remaining: 41s
330:	test: 0.8945642 test1: 0.8823283	best: 0.8823283 (330)	total: 20s	remaining: 40.4s
340:	test: 0.8948307 test1: 0.8825061	best: 0.8825066 (339)	total: 20.5s	remaining: 39.7s
350:	test: 0.8950636 test1: 0.8827360	best: 0.8827360 (350)	total: 21.1s	remaining: 39s
360:	test: 0.8953873 test1: 0.8830067	best: 0.8830067 (360)	total: 21.6s	remaining: 38.3s
370:	test: 0.8957220 test1: 0.8831639	best: 0.8831700 (369)	total: 22.2s	remaining: 37.6s
380:	test: 0.8959290 test1: 0.8833008	best: 0.8833008 (380)	total: 22.7s	remaining: 37s
390:	test: 0.8962096 test1: 0.8835008	best: 0.8835008 (390)	total: 23.3s	remaining: 36.3s
400:	test: 0.8965564 test1: 0.8838152	best: 0.8838152 (400)	total: 23.9s	remaining: 35.6s
410:	test: 0.8968402 test1: 0.8839956	best: 0.8839956 (410)	total: 24.4s	remaining: 35s
420:	test: 0.8971198 test1: 0.8841992	best: 0.8841993 (418)	total: 25s	remaining: 34.4s
430:	test: 0.8972062 test1: 0.8842719	best: 0.8842733 (426)	total: 25.5s	remaining: 33.7s
440:	test: 0.8972892 test1: 0.8842801	best: 0.8842804 (439)	total: 26.1s	remaining: 33.1s
450:	test: 0.8973008 test1: 0.8842844	best: 0.8842896 (442)	total: 26.6s	remaining: 32.4s
460:	test: 0.8973092 test1: 0.8842851	best: 0.8842896 (442)	total: 27.1s	remaining: 31.7s
470:	test: 0.8973149 test1: 0.8842831	best: 0.8842896 (442)	total: 27.7s	remaining: 31.1s
480:	test: 0.8973202 test1: 0.8842811	best: 0.8842896 (442)	total: 28.2s	remaining: 30.4s
490:	test: 0.8973266 test1: 0.8842789	best: 0.8842896 (442)	total: 28.7s	remaining: 29.8s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8842896115
bestIteration = 442

Shrink model to first 443 iterations.
```

Out[133]:
```
<catboost.core.CatBoostClassifier at 0x4e4ca28bb0>
```

Задание 0:

- bestTest = 0.8827161236
- bestIteration = 419

Задание 4:

- bestTest = 0.8842896115
- bestIteration = 442

Вывод:

- Добавление новых признаков (Задание 4) незначительно улучшило 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 [135]:
```
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 [136]:
```
temp = x_train_task_5.groupby('card1')['D15'].agg(['mean']).rename({'mean':'D15_card1_mean'},axis=1)
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'},axis=1)
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'},axis=1)
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'},axis=1)
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'},axis=1)
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'},axis=1)
x_train_task_5 = pd.merge(x_train_task_5,temp,on='card6',how='left')
```

В [137]:
```
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',how='left')
```

В [138]:
```
temp = x_test_task_5.groupby('card1')['D15'].agg(['mean']).rename({'mean':'D15_card1_mean'},axis=1)
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'},axis=1)
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'},axis=1)
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'},axis=1)
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'},axis=1)
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'},axis=1)
x_test_task_5 = pd.merge(x_test_task_5,temp,on='card6',how='left')
```

В [139]:
```
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='left')
```

В [140]:
```
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 [141]:
```
### 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 [142]:
```
eval_sets= [
    (x_train_task_5[xgb_numerical_features + categorical_features], y_train),
    (x_test_task_5[xgb_numerical_features + categorical_features], y_test)
]
```

В [143]:
```
cb_model.fit(
    x_train_task_5[xgb_numerical_features + categorical_features],
    y_train,
    ### cat_features = categorical_features,
    eval_set=eval_sets)
```

```
0:      test: 0.6668196 test1: 0.6364106       best: 0.6364106 (0)     total: 226ms    remaining: 3m 45s
10:     test: 0.7837131 test1: 0.7731993       best: 0.7744707 (8)     total: 1.22s    remaining: 1m 50s
20:     test: 0.8208458 test1: 0.8134966       best: 0.8134966 (20)    total: 1.82s    remaining: 1m 24s
30:     test: 0.8484551 test1: 0.8388704       best: 0.8388704 (30)    total: 2.39s    remaining: 1m 14s
40:     test: 0.8539346 test1: 0.8452090       best: 0.8452090 (40)    total: 2.98s    remaining: 1m 9s
50:     test: 0.8556360 test1: 0.8466824       best: 0.8467786 (49)    total: 3.55s    remaining: 1m 6s
60:     test: 0.8594347 test1: 0.8502323       best: 0.8507420 (58)    total: 4.15s    remaining: 1m 3s
70:     test: 0.8611161 test1: 0.8513599       best: 0.8520064 (65)    total: 4.76s    remaining: 1m 2s
80:     test: 0.8641617 test1: 0.8540539       best: 0.8540607 (78)    total: 5.39s    remaining: 1m 1s
90:     test: 0.8652170 test1: 0.8542878       best: 0.8544052 (81)    total: 5.99s    remaining: 59.8s
100:    test: 0.8681806 test1: 0.8576883       best: 0.8577066 (98)    total: 6.64s    remaining: 59.1s
110:    test: 0.8693815 test1: 0.8586095       best: 0.8586703 (109)   total: 7.22s    remaining: 57.8s
120:    test: 0.8706474 test1: 0.8600285       best: 0.8600285 (120)   total: 7.82s    remaining: 56.8s
130:    test: 0.8723017 test1: 0.8615211       best: 0.8615473 (129)   total: 8.4s     remaining: 55.7s
140:    test: 0.8742607 test1: 0.8638919       best: 0.8638919 (140)   total: 8.99s    remaining: 54.8s
150:    test: 0.8764717 test1: 0.8653649       best: 0.8653649 (150)   total: 9.59s    remaining: 53.9s
160:    test: 0.8779948 test1: 0.8673450       best: 0.8673450 (160)   total: 10.2s    remaining: 53.1s
170:    test: 0.8800262 test1: 0.8692304       best: 0.8692304 (170)   total: 10.8s    remaining: 52.4s
180:    test: 0.8815406 test1: 0.8706230       best: 0.8706230 (180)   total: 11.4s    remaining: 51.6s
190:    test: 0.8826109 test1: 0.8713968       best: 0.8713968 (190)   total: 12.1s    remaining: 51.2s
200:    test: 0.8833512 test1: 0.8719780       best: 0.8719780 (200)   total: 12.7s    remaining: 50.5s
210:    test: 0.8847313 test1: 0.8737443       best: 0.8737563 (209)   total: 13.3s    remaining: 49.8s
220:    test: 0.8855477 test1: 0.8744963       best: 0.8744963 (220)   total: 13.9s    remaining: 49s
230:    test: 0.8862676 test1: 0.8751748       best: 0.8751748 (230)   total: 14.5s    remaining: 48.2s
240:    test: 0.8872596 test1: 0.8760118       best: 0.8760118 (240)   total: 15.1s    remaining: 47.5s
250:    test: 0.8878350 test1: 0.8763940       best: 0.8763940 (250)   total: 15.7s    remaining: 46.8s
260:    test: 0.8884866 test1: 0.8769061       best: 0.8769061 (260)   total: 16.3s    remaining: 46.1s
270:    test: 0.8889540 test1: 0.8772844       best: 0.8772844 (270)   total: 16.9s    remaining: 45.4s
280:    test: 0.8895295 test1: 0.8778524       best: 0.8778524 (280)   total: 17.4s    remaining: 44.6s
290:    test: 0.8900237 test1: 0.8782163       best: 0.8782163 (290)   total: 18s      remaining: 43.9s
300:    test: 0.8905677 test1: 0.8786654       best: 0.8786654 (300)   total: 18.6s    remaining: 43.2s
310:    test: 0.8908104 test1: 0.8788436       best: 0.8788453 (307)   total: 19.1s    remaining: 42.4s
320:    test: 0.8910466 test1: 0.8789826       best: 0.8790020 (317)   total: 19.7s    remaining: 41.7s
330:    test: 0.8914842 test1: 0.8793535       best: 0.8793535 (330)   total: 20.3s    remaining: 41s
340:    test: 0.8919452 test1: 0.8797271       best: 0.8797299 (339)   total: 20.9s    remaining: 40.4s
350:    test: 0.8922958 test1: 0.8800598       best: 0.8800631 (347)   total: 21.5s    remaining: 39.7s
360:    test: 0.8928456 test1: 0.8804676       best: 0.8804753 (359)   total: 22.1s    remaining: 39.1s
370:    test: 0.8933041 test1: 0.8806523       best: 0.8806523 (370)   total: 22.7s    remaining: 38.4s
380:    test: 0.8938593 test1: 0.8810213       best: 0.8810213 (380)   total: 23.3s    remaining: 37.8s
390:    test: 0.8943036 test1: 0.8814521       best: 0.8814521 (390)   total: 23.9s    remaining: 37.2s
400:    test: 0.8948531 test1: 0.8818168       best: 0.8818168 (400)   total: 24.5s    remaining: 36.5s
410:    test: 0.8953730 test1: 0.8822657       best: 0.8822657 (410)   total: 25.1s    remaining: 35.9s
420:    test: 0.8959407 test1: 0.8827536       best: 0.8827536 (420)   total: 25.9s    remaining: 35.7s
430:    test: 0.8963353 test1: 0.8830607       best: 0.8830607 (430)   total: 26.9s    remaining: 35.5s
440:    test: 0.8964688 test1: 0.8832173       best: 0.8832173 (440)   total: 27.8s    remaining: 35.3s
450:    test: 0.8965020 test1: 0.8832485       best: 0.8832485 (450)   total: 28.4s    remaining: 34.6s
460:    test: 0.8965108 test1: 0.8832492       best: 0.8832492 (460)   total: 28.9s    remaining: 33.8s
470:    test: 0.8965165 test1: 0.8832444       best: 0.8832495 (463)   total: 29.4s    remaining: 33s
480:    test: 0.8965244 test1: 0.8832453       best: 0.8832495 (463)   total: 29.9s    remaining: 32.3s
490:    test: 0.8965325 test1: 0.8832451       best: 0.8832495 (463)   total: 30.4s    remaining: 31.6s
500:    test: 0.8965381 test1: 0.8832435       best: 0.8832495 (463)   total: 31s      remaining: 30.8s
510:    test: 0.8965444 test1: 0.8832421       best: 0.8832495 (463)   total: 31.5s    remaining: 30.1s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8832494667
bestIteration = 463

Shrink model to first 464 iterations.
```

Out[143]: <catboost.core.CatBoostClassifier at 0x4e4ca28bb0>

Задание 0:

- bestTest = 0.8827161236
- bestIteration = 419

Задание 5:

- bestTest = 0.8832494667
- bestIteration = 463

Вывод:

- Добавление новых признаков (Задание 5) незначительно улучшило качества модели по сравнению с базовым решением.

Задание 6:

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

В [91]:
```
import math
# print(5.1 - int(5.1))
# x = math.modf(3.456)
# print(x[0])
# print(x[1])
```

В [92]:
```
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 [93]:
```
import math
print(math.modf(45.8978))

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

```
(0.8977999999999966, 45.0)
```

В [94]:
```
# 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 [95]:
```
# x_train_task_6['TransactionAmr_log'] = zip(*x_train_task_6['TransactionAmt'].map(function_log))
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 [96]:
```
task6_features = [
 'TransactionAmr_intager',
 'TransactionAmr_fractional',
 'TransactionAmr_log',
]
```

В [150]:
```
#x_train_task_3["TransactionAmt"]
```

В [149]:
```
x_train_task_6[task6_features]
```

Out[149]:

	TransactionAmr_intager	TransactionAmr_fractional	TransactionAmr_log
141582	218.0	0.000000	5.384495
131503	50.0	0.000000	3.912023
173925	77.0	0.000000	4.343805
177012	57.0	0.950001	4.059581
69958	44.0	0.000000	3.784190
...	...	...	...
4848	25.0	0.000000	3.218876
14879	40.0	0.000000	3.688879
36680	24.0	0.000000	3.178054
118456	63.0	0.950001	4.158102
5139	59.0	0.000000	4.077538

135000 rows × 3 columns

В [98]:
```
# eval_sets= [
#     (x_train_task_6[xgb_numerical_features + categorical_features], y_train),
#     (x_test_task_6[xgb_numerical_features + categorical_features], y_test)
# ]
```

В [99]:
```
# cb_model.fit(
#     x_train_task_6[xgb_numerical_features + task6_features],
#     y_train,
#     cat_features = categorical_features,
#     eval_set=eval_sets)
```

В [100]:
```
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 [101]:
```
# cb_model = cb.CatBoostClassifier(**cb_params)
# cb_model.fit(x_train_task_6[xgb_numerical_features + task6_features], y_train, eval_set=eval_sets)
```

В [102]:
```
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)     total: 165ms    remaining: 2m 44s
10:     test: 0.7729142 test1: 0.7622869       best: 0.7622965 (9)     total: 1.39s    remaining: 2m 4s
20:     test: 0.8279478 test1: 0.8231270       best: 0.8231270 (20)    total: 2.58s    remaining: 2m
30:     test: 0.8414742 test1: 0.8332268       best: 0.8332268 (30)    total: 3.47s    remaining: 1m 48s
40:     test: 0.8490078 test1: 0.8425562       best: 0.8427865 (39)    total: 4.06s    remaining: 1m 34s
50:     test: 0.8524722 test1: 0.8458381       best: 0.8458381 (50)    total: 4.64s    remaining: 1m 26s
60:     test: 0.8575901 test1: 0.8498734       best: 0.8499933 (57)    total: 5.21s    remaining: 1m 20s
70:     test: 0.8582895 test1: 0.8501271       best: 0.8511148 (64)    total: 5.81s    remaining: 1m 16s
80:     test: 0.8625141 test1: 0.8538566       best: 0.8538566 (80)    total: 6.38s    remaining: 1m 12s
90:     test: 0.8654997 test1: 0.8570493       best: 0.8570493 (90)    total: 6.96s    remaining: 1m 9s
100:    test: 0.8685545 test1: 0.8605576       best: 0.8605576 (100)   total: 7.56s    remaining: 1m 7s
110:    test: 0.8696675 test1: 0.8610154       best: 0.8610279 (106)   total: 8.13s    remaining: 1m 5s
120:    test: 0.8713190 test1: 0.8625735       best: 0.8625735 (120)   total: 8.71s    remaining: 1m 3s
130:    test: 0.8732775 test1: 0.8646900       best: 0.8646900 (130)   total: 9.29s    remaining: 1m 1s
140:    test: 0.8747180 test1: 0.8658132       best: 0.8658132 (140)   total: 9.88s    remaining: 1m
150:    test: 0.8761941 test1: 0.8672323       best: 0.8672323 (150)   total: 10.4s    remaining: 58.7s
160:    test: 0.8778482 test1: 0.8687972       best: 0.8687972 (160)   total: 11s      remaining: 57.5s
170:    test: 0.8797394 test1: 0.8709394       best: 0.8709510 (169)   total: 11.6s    remaining: 56.3s
180:    test: 0.8810733 test1: 0.8723657       best: 0.8723657 (180)   total: 12.2s    remaining: 55.4s
190:    test: 0.8822296 test1: 0.8738214       best: 0.8738214 (190)   total: 12.8s    remaining: 54.3s
200:    test: 0.8831831 test1: 0.8745627       best: 0.8745627 (200)   total: 13.4s    remaining: 53.4s
210:    test: 0.8837338 test1: 0.8748745       best: 0.8749198 (206)   total: 14s      remaining: 52.5s
220:    test: 0.8848292 test1: 0.8756852       best: 0.8756852 (220)   total: 14.6s    remaining: 51.6s
230:    test: 0.8857132 test1: 0.8767333       best: 0.8767333 (230)   total: 15.2s    remaining: 50.6s
240:    test: 0.8860846 test1: 0.8772052       best: 0.8772052 (240)   total: 15.7s    remaining: 49.6s
250:    test: 0.8866281 test1: 0.8779573       best: 0.8779573 (250)   total: 16.3s    remaining: 48.7s
260:    test: 0.8873535 test1: 0.8784585       best: 0.8784585 (260)   total: 16.9s    remaining: 47.9s
270:    test: 0.8880070 test1: 0.8789357       best: 0.8789396 (269)   total: 17.5s    remaining: 47.1s
280:    test: 0.8885363 test1: 0.8792828       best: 0.8792828 (280)   total: 18.1s    remaining: 46.2s
290:    test: 0.8891513 test1: 0.8796245       best: 0.8796245 (290)   total: 18.6s    remaining: 45.4s
300:    test: 0.8895779 test1: 0.8798824       best: 0.8798824 (300)   total: 19.2s    remaining: 44.6s
310:    test: 0.8899194 test1: 0.8801941       best: 0.8801941 (310)   total: 19.8s    remaining: 43.8s
320:    test: 0.8902665 test1: 0.8805143       best: 0.8805143 (320)   total: 20.3s    remaining: 43s
330:    test: 0.8906451 test1: 0.8808342       best: 0.8808342 (330)   total: 20.9s    remaining: 42.2s
340:    test: 0.8910895 test1: 0.8811072       best: 0.8811072 (340)   total: 21.5s    remaining: 41.5s
350:    test: 0.8913818 test1: 0.8812377       best: 0.8812377 (350)   total: 22s      remaining: 40.8s
360:    test: 0.8918047 test1: 0.8815479       best: 0.8815479 (360)   total: 22.6s    remaining: 40s
370:    test: 0.8921678 test1: 0.8817415       best: 0.8817415 (370)   total: 23.2s    remaining: 39.3s
380:    test: 0.8926238 test1: 0.8820942       best: 0.8821052 (379)   total: 23.7s    remaining: 38.6s
390:    test: 0.8928945 test1: 0.8823416       best: 0.8823416 (390)   total: 24.3s    remaining: 37.8s
400:    test: 0.8931708 test1: 0.8825326       best: 0.8825348 (397)   total: 24.8s    remaining: 37.1s
410:    test: 0.8933258 test1: 0.8826740       best: 0.8826762 (409)   total: 25.3s    remaining: 36.3s
420:    test: 0.8935168 test1: 0.8828081       best: 0.8828082 (419)   total: 26s      remaining: 35.7s
430:    test: 0.8935877 test1: 0.8828855       best: 0.8828855 (430)   total: 26.9s    remaining: 35.6s
440:    test: 0.8935944 test1: 0.8828839       best: 0.8828861 (437)   total: 28.1s    remaining: 35.6s
450:    test: 0.8936109 test1: 0.8828873       best: 0.8828945 (443)   total: 29s      remaining: 35.3s
460:    test: 0.8936144 test1: 0.8828809       best: 0.8828945 (443)   total: 29.8s    remaining: 34.8s
470:    test: 0.8936352 test1: 0.8828844       best: 0.8828945 (443)   total: 30.4s    remaining: 34.1s
480:    test: 0.8936506 test1: 0.8828790       best: 0.8828945 (443)   total: 30.9s    remaining: 33.3s
490:    test: 0.8936570 test1: 0.8828777       best: 0.8828945 (443)   total: 31.4s    remaining: 32.5s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8828945346
bestIteration = 443

Shrink model to first 444 iterations.
```

Out[102]:
```
<catboost.core.CatBoostClassifier at 0x4e4ca28bb0>
```

Задание 0 (без обработки):

- bestTest = 0.8827161236
- bestIteration = 419

Задание 6:

- bestTest = 0.8828945346
- bestIteration = 443

Вывод:

- Добавление новых признаков (Задание 6) незначительно улучшило 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.

См. "Урок 4 Предварительная обработка признаков/Категориальные признаки/Второй способ". Файл webinar4_features_part1.ipynb.

В [103]:
```
x_train_task_7 = []
x_test_task_7 = []
x_train_task_7 = x_train.copy()
x_test_task_7 = x_test.copy()
```

В [158]:
```
data = []
data_test = []
data = x_train_task_1.copy()
data_test = x_test_task_1.copy()
```

В [159]:
```
x_train_task_7[['P_emaildomain', 'R_emaildomain']]
```

Out[159]:

	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 [160]:
```
x_test_task_7[['P_emaildomain', 'R_emaildomain']]
```

Out[160]:

	P_emaildomain	R_emaildomain
78715	anonymous.com	Unknown
907	comcast.net	Unknown
87782	gmail.com	gmail.com
55343	hotmail.com	Unknown
7372	anonymous.com	Unknown
...	...	...
4018	yahoo.com	Unknown
79718	gmail.com	anonymous.com
23131	aol.com	Unknown
99884	hotmail.com	hotmail.com
168530	aol.com	yahoo.com

45000 rows × 2 columns

В [161]:
```
freq_encoder = data["P_emaildomain"].value_counts(normalize=True)
data["P_emaildomain_freq_enc"] = data["P_emaildomain"].map(freq_encoder)
freq_encoder = data["R_emaildomain"].value_counts(normalize=True)
data["R_emaildomain_freq_enc"] = data["R_emaildomain"].map(freq_encoder)
```

В [162]:
```
freq_encoder = data_test["P_emaildomain"].value_counts(normalize=True)
data_test["P_emaildomain_freq_enc"] = data_test["P_emaildomain"].map(freq_encoder)
freq_encoder = data_test["R_emaildomain"].value_counts(normalize=True)
data_test["R_emaildomain_freq_enc"] = data_test["R_emaildomain"].map(freq_encoder)
```

В [163]:
```
data[["P_emaildomain", "P_emaildomain_freq_enc", "R_emaildomain", "R_emaildomain_freq_enc"]]
```

Out[163]:

	P_emaildomain	P_emaildomain_freq_enc	R_emaildomain	R_emaildomain_freq_enc
141582	Unknown	0.158096	Unknown	0.665281
131503	yahoo.com	0.160852	yahoo.com	0.031067
173925	Unknown	0.158096	Unknown	0.665281
177012	aol.com	0.048037	Unknown	0.665281
69958	Unknown	0.158096	Unknown	0.665281
...	...	...	...	...
4848	anonymous.com	0.073985	Unknown	0.665281
14879	Unknown	0.158096	anonymous.com	0.054837
36680	Unknown	0.158096	Unknown	0.665281
118456	gmail.com	0.373281	Unknown	0.665281
5139	yahoo.com	0.160852	Unknown	0.665281

135000 rows × 4 columns

В [164]:
```
categorical_features = [
 'P_emaildomain_freq_enc',
 'R_emaildomain_freq_enc'
]
```

В [165]:
```
data[categorical_features] = data[categorical_features].astype(str)
data_test[categorical_features] = data_test[categorical_features].astype(str)
```

В [166]:
```
eval_sets= [
    (data[xgb_numerical_features + categorical_features], y_train),
    (data_test[xgb_numerical_features + categorical_features], y_test)
]
```

В [167]:
```
cb_model.fit(
    data[xgb_numerical_features + categorical_features],
    y_train,
    #cat_features = categorical_features,
    eval_set=eval_sets)
```

```
0:      test: 0.6426443 test1: 0.6397917        best: 0.6397917 (0)     total: 173ms    remaining: 2m 52s
10:     test: 0.7842472 test1: 0.7762249        best: 0.7762864 (9)     total: 1.08s    remaining: 1m 37s
20:     test: 0.8205347 test1: 0.8149997        best: 0.8152477 (19)    total: 1.88s    remaining: 1m 27s
30:     test: 0.8417268 test1: 0.8334874        best: 0.8334874 (30)    total: 2.65s    remaining: 1m 22s
40:     test: 0.8489144 test1: 0.8393514        best: 0.8393800 (38)    total: 3.33s    remaining: 1m 17s
50:     test: 0.8552671 test1: 0.8471204        best: 0.8472737 (48)    total: 3.9s     remaining: 1m 12s
60:     test: 0.8553998 test1: 0.8466190        best: 0.8472958 (53)    total: 4.48s    remaining: 1m 8s
70:     test: 0.8579710 test1: 0.8487815        best: 0.8491857 (68)    total: 5.07s    remaining: 1m 6s
80:     test: 0.8619628 test1: 0.8519043        best: 0.8519043 (80)    total: 5.66s    remaining: 1m 4s
90:     test: 0.8633333 test1: 0.8526949        best: 0.8527723 (89)    total: 6.25s    remaining: 1m 2s
100:    test: 0.8659497 test1: 0.8554849        best: 0.8554849 (100)   total: 6.81s    remaining: 1m
110:    test: 0.8682495 test1: 0.8576799        best: 0.8577087 (109)   total: 7.38s    remaining: 59.1s
120:    test: 0.8708602 test1: 0.8608080        best: 0.8609317 (119)   total: 7.96s    remaining: 57.8s
130:    test: 0.8723963 test1: 0.8626150        best: 0.8626150 (130)   total: 8.54s    remaining: 56.7s
140:    test: 0.8748761 test1: 0.8650605        best: 0.8650605 (140)   total: 9.15s    remaining: 55.8s
150:    test: 0.8770454 test1: 0.8670789        best: 0.8670789 (150)   total: 9.78s    remaining: 55s
160:    test: 0.8777726 test1: 0.8673882        best: 0.8674358 (158)   total: 10.4s    remaining: 54s
170:    test: 0.8795917 test1: 0.8690174        best: 0.8690174 (170)   total: 11s      remaining: 53.1s
180:    test: 0.8811203 test1: 0.8704157        best: 0.8704157 (180)   total: 11.6s    remaining: 52.3s
190:    test: 0.8829921 test1: 0.8724565        best: 0.8724565 (190)   total: 12.1s    remaining: 51.4s
200:    test: 0.8839487 test1: 0.8731294        best: 0.8731294 (200)   total: 12.7s    remaining: 50.6s
210:    test: 0.8850984 test1: 0.8745328        best: 0.8745381 (209)   total: 13.3s    remaining: 49.8s
220:    test: 0.8863106 test1: 0.8758305        best: 0.8758683 (218)   total: 13.9s    remaining: 49s
230:    test: 0.8878623 test1: 0.8771902        best: 0.8771969 (229)   total: 14.5s    remaining: 48.2s
240:    test: 0.8885018 test1: 0.8777146        best: 0.8777146 (240)   total: 15.1s    remaining: 47.4s
250:    test: 0.8895272 test1: 0.8789737        best: 0.8789737 (250)   total: 15.6s    remaining: 46.6s
260:    test: 0.8903552 test1: 0.8797177        best: 0.8797177 (260)   total: 16.2s    remaining: 45.9s
270:    test: 0.8911732 test1: 0.8806420        best: 0.8806420 (270)   total: 16.8s    remaining: 45.1s
280:    test: 0.8919080 test1: 0.8812047        best: 0.8812047 (280)   total: 17.3s    remaining: 44.4s
290:    test: 0.8924629 test1: 0.8816749        best: 0.8816749 (290)   total: 17.9s    remaining: 43.6s
300:    test: 0.8928405 test1: 0.8820332        best: 0.8820332 (300)   total: 18.4s    remaining: 42.8s
310:    test: 0.8932946 test1: 0.8823053        best: 0.8823053 (310)   total: 19s      remaining: 42.2s
320:    test: 0.8938356 test1: 0.8828631        best: 0.8828631 (320)   total: 19.6s    remaining: 41.5s
330:    test: 0.8944465 test1: 0.8833127        best: 0.8833127 (330)   total: 20.2s    remaining: 40.8s
340:    test: 0.8948192 test1: 0.8836734        best: 0.8836734 (340)   total: 20.7s    remaining: 40.1s
350:    test: 0.8952297 test1: 0.8840542        best: 0.8840558 (348)   total: 21.3s    remaining: 39.3s
360:    test: 0.8957246 test1: 0.8843318        best: 0.8843398 (359)   total: 21.8s    remaining: 38.7s
370:    test: 0.8959591 test1: 0.8844523        best: 0.8844523 (370)   total: 22.4s    remaining: 37.9s
380:    test: 0.8965268 test1: 0.8849417        best: 0.8849417 (380)   total: 22.9s    remaining: 37.3s
390:    test: 0.8968244 test1: 0.8851769        best: 0.8851771 (389)   total: 23.5s    remaining: 36.6s
400:    test: 0.8970284 test1: 0.8854738        best: 0.8854738 (400)   total: 24s      remaining: 35.9s
410:    test: 0.8972017 test1: 0.8856085        best: 0.8856154 (409)   total: 24.6s    remaining: 35.2s
420:    test: 0.8973978 test1: 0.8858786        best: 0.8858786 (420)   total: 25.1s    remaining: 34.5s
430:    test: 0.8974557 test1: 0.8859049        best: 0.8859049 (430)   total: 25.7s    remaining: 33.9s
440:    test: 0.8974649 test1: 0.8859063        best: 0.8859063 (437)   total: 26.2s    remaining: 33.3s
450:    test: 0.8974734 test1: 0.8859078        best: 0.8859092 (446)   total: 26.8s    remaining: 32.6s
460:    test: 0.8974830 test1: 0.8859084        best: 0.8859097 (458)   total: 27.3s    remaining: 31.9s
470:    test: 0.8974859 test1: 0.8859028        best: 0.8859097 (458)   total: 27.8s    remaining: 31.2s
480:    test: 0.8974898 test1: 0.8858989        best: 0.8859097 (458)   total: 28.3s    remaining: 30.6s
490:    test: 0.8974926 test1: 0.8858914        best: 0.8859097 (458)   total: 28.8s    remaining: 29.9s
500:    test: 0.8974958 test1: 0.8858870        best: 0.8859097 (458)   total: 29.3s    remaining: 29.2s
Stopped by overfitting detector  (50 iterations wait)

bestTest = 0.8859097396
bestIteration = 458

Shrink model to first 459 iterations.
```

Out[167]: `<catboost.core.CatBoostClassifier at 0x4e4ca28bb0>`

Вывод:

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 = 0.8842896115
- bestIteration = 442

Вывод:

- Добавление новых признаков (Задание 4) незначительно улучшило качества модели по сравнению с базовым решением.

Задание 5:

- bestTest = 0.8832494667
- bestIteration = 463

Вывод:

- Добавление новых признаков (Задание 5) незначительно улучшило качества модели по сравнению с базовым решением.

Задание 6:

- bestTest = 0.8828945346
- bestIteration = 443

Вывод:

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

Задание 7:

- bestTest = 0.8859097396
- bestIteration = 458

Вывод:

- Добавление новых признаков (Задание 7) улучшило качества модели по сравнению с базовым решением.

В [ ]:

```

```