Earlier My score was

Below score using LightLbm Model

Now it is

https://www.youtube.com/watch?v=DfZpljN9BYU&t=2042s

Helped me to understand datasets.

Based on this I did required changes on my model.

Outlier handle

As is evident from multiple discussions:¶ https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/73571 https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/73922 https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/73024 The 2207 rows with target values < -33 do seem to have an important role in this synthesized dataset.

So its very important for us to handle outlier carefully. Even i have seen when i handled outlier my kaggle score improved alot

Added new features holiday features

This heavily borrowed from Chau Ngoc Huynh's https://www.kaggle.com/chauhuynh/my-first-kernel-3-699. I try to create some features using workalendar, which was suggested on Bojan Tunguz's post (https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/74052) by Kjetil Åmdal-Sævik.¶

I have created new features in the train and test sets based on the number of working days (in the Brazilian calendar) between the first_active_month and the 8 major national holidays.This holiday features improved alot my score

Additional features which i borrowed from https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/82107 https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/82055

elapsed_time ,total purshase amount and so many features idea i borrowed from above discussion

Combine model both with outlier and without outlier.

Got idea from below

The basic idea behind this is:

Training model without outliers make the model more accurate for non-outliers. A great proportion of the error is caused by outliers, so we need to use a model training with outliers to predict them. How to find them out? build a classifier!

```
Image(filename='/content/drive/MyDrive/elo-merchant-category-recommendation/20210302_140807.j
```

LightLbm Model

I played with Xgboost,catboost,esmblemed model. I hypertuned it tried to managed it with my low configure system. Even these models did not provide better result ,compare to lightlbm.

**Observation **

Faster training speed and higher efficiency.

Lower memory usage.

Better accuracy.

Support of parallel and GPU learning.

Capable of handling large-scale data.


```
# Import all dependencies
import pandas as pd
import numpy as np
from sklearn import preprocessing
import warnings
import datetime
import datetime
from datetime import date, datetime

from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import numpy as np
import pandas as pd
import datetime
import gc
import matplotlib.pyplot as plt
import seaborn as sns
import lightgbm as lgb
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings('ignore')
np.random.seed(4590)
#https://www.kaggle.com/raddar/card-id-loyalty-different-points-in-time

from workalendar.america import Brazil
from IPython.display import Image
import pickle



!pip3 install workalendar
```

```
!pip3 install workalendar
```

```
Requirement already satisfied: workalendar in /usr/local/lib/python3.7/dist-packages (1
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: backports.zoneinfo; python_version < "3.9" in /usr/local
Requirement already satisfied: skyfield-data in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyluach in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: lunardate in /usr/local/lib/python3.7/dist-packages (fro
Requirement already satisfied: skyfield in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: pyCalverter in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: setuptools>=1.0 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: jplephem>=2.13 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from sk
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: sgp4>=2.2 in /usr/local/lib/python3.7/dist-packages (fro
```

```
import os
import gc

warnings.filterwarnings("ignore")



from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
```

Double-click (or enter) to edit

```
#Reduce the memory usage - Inspired by Panchajanya Banerjee
def reduce_mem_usage(df, verbose=True):
    numerics = ['int16', 'int32', 'int64', 'float16', 'float32', 'float64']
    start_mem = df.memory_usage().sum() / 1024**2
    for col in df.columns:
        col_type = df[col].dtypes
        if col_type in numerics:
            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:
```

```
                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.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif 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)
    end_mem = df.memory_usage().sum() / 1024**2
    if verbose: print('Mem. usage decreased to {:5.2f} Mb ({:.1f}% reduction)'.format(end_mem
    return df
```

```
new_trans = pd.read_csv('/content/drive/MyDrive/elo-merchant-category-recommendation/new_merc
```

```
new_trans.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1963031 entries, 0 to 1963030
Data columns (total 14 columns):
 #   Column                Dtype  
---  ------                -----  
 0   authorized_flag       object 
 1   card_id               object 
 2   city_id               int64  
 3   category_1            object 
 4   installments          int64  
 5   category_3            object 
 6   merchant_category_id  int64  
 7   merchant_id           object 
 8   month_lag             int64  
 9   purchase_amount       float64
 10  purchase_date         object 
 11  category_2            float64
 12  state_id              int64  
 13  subsector_id          int64  
dtypes: float64(2), int64(6), object(6)
memory usage: 209.7+ MB
```

```
hist_trans = pd.read_csv('/content/drive/MyDrive/elo-merchant-category-recommendation/histori
```

```
train = pd.read_csv("/content/drive/MyDrive/elo-merchant-category-recommendation/train.csv")
```

```
test = pd.read_csv("/content/drive/MyDrive/elo-merchant-category-recommendation/test.csv")
```

```
train=reduce_mem_usage(train)
```

```
Mem. usage decreased to  4.04 Mb (56.2% reduction)
```

```
test==reduce_mem_usage(test)
```

```
Mem. usage decreased to  2.24 Mb (52.5% reduction)
```

	first_active_month	card_id	feature_1	feature_2	feature_3
0	True	True	True	True	True
1	True	True	True	True	True
2	True	True	True	True	True
3	True	True	True	True	True
4	True	True	True	True	True
...	...	...	...	...	...
123618	True	True	True	True	True
123619	True	True	True	True	True
123620	True	True	True	True	True
123621	True	True	True	True	True
123622	True	True	True	True	True

123623 rows × 5 columns

```
hist_trans=reduce_mem_usage(hist_trans)
```

```
Mem. usage decreased to 1749.11 Mb (43.7% reduction)
```

```
new_trans=reduce_mem_usage(new_trans)
```

```
Mem. usage decreased to 114.20 Mb (45.5% reduction)
```

```
def missingvalue(df):
    for i in df.columns:
        if df[i].dtype == "object":
            df[i] = df[i].fillna("other")
        elif (df[i].dtype == "int64" or df[i].dtype == "float64"):
            df[i] = df[i].fillna(df[i].mean())
        else:
            pass
    return df

# missing values for all datasets
```

```
for df in [train, test,new_trans,  hist_trans]:
    missingvalue(df)


train=train.drop_duplicates(subset=['target'])


for df in [hist_trans,new_trans]:
    df['category_2'].fillna(1.0,inplace=True)
    df['category_3'].fillna('A',inplace=True)
    df['merchant_id'].fillna('M_ID_00a6ca8a8a',inplace=True)


def get_new_columns(name,aggs):
    return [name + '_' + k + '_' + agg for k in aggs.keys() for agg in aggs[k]]


#https://stackoverflow.com/questions/33817380/workalendar-europe-library-how-to-find-holidays
cal = Brazil()
for yr in [2011,2012,2013,2014,2015,2016,2017]:
    print(yr,cal.holidays(yr))
```

```
 2011 [(datetime.date(2011, 1, 1), 'New year'), (datetime.date(2011, 4, 21), "Tiradentes
 2012 [(datetime.date(2012, 1, 1), 'New year'), (datetime.date(2012, 4, 8), 'Easter Sund
 2013 [(datetime.date(2013, 1, 1), 'New year'), (datetime.date(2013, 3, 31), 'Easter Sun
 2014 [(datetime.date(2014, 1, 1), 'New year'), (datetime.date(2014, 4, 20), 'Easter Sun
 2015 [(datetime.date(2015, 1, 1), 'New year'), (datetime.date(2015, 4, 5), 'Easter Sund
 2016 [(datetime.date(2016, 1, 1), 'New year'), (datetime.date(2016, 3, 27), 'Easter Sun
 2017 [(datetime.date(2017, 1, 1), 'New year'), (datetime.date(2017, 4, 16), 'Easter Sun
```

```
cal.holidays(2013)[1]
```

```
 (datetime.date(2013, 3, 31), 'Easter Sunday')
```

```
for df in [hist_trans,new_trans]:
    df['purchase_date'] = pd.to_datetime(df['purchase_date'])
    df['year'] = df['purchase_date'].dt.year
    df['weekofyear'] = df['purchase_date'].dt.weekofyear
    df['month'] = df['purchase_date'].dt.month
    df['dayofweek'] = df['purchase_date'].dt.dayofweek
    df['weekend'] = (df.purchase_date.dt.weekday >=5).astype(int)
    df['hour'] = df['purchase_date'].dt.hour
    df['authorized_flag'] = df['authorized_flag'].map({'Y':1, 'N':0})
    df['category_1'] = df['category_1'].map({'Y':1, 'N':0})

    df['month_diff'] = ((datetime.datetime.today() - df['purchase_date']).dt.days)//30
    df['month_diff'] += df['month_lag']


aggs = {}
for col in ['purchase_amount','month','hour','weekofyear','dayofweek','year','subsector_id','
    aggs[col] = ['nunique']
```

```
    aggs[col] = ['nunique']

aggs['purchase_amount'] = ['sum','max','min','mean','var']
aggs['installments'] = ['sum','max','min','mean','var']
aggs['purchase_date'] = ['max','min']
aggs['month_lag'] = ['max','min','mean','var']
aggs['month_diff'] = ['mean']
aggs['authorized_flag'] = ['sum', 'mean']
aggs['weekend'] = ['sum', 'mean']
aggs['category_1'] = ['sum', 'mean']
aggs['card_id'] = ['size']

for col in ['category_2','category_3']:
    hist_trans[col+'_mean'] =hist_trans.groupby([col])['purchase_amount'].transform('mean')
    aggs[col+'_mean'] = ['mean']

new_columns = get_new_columns('hist',aggs)
hist_trans_group = hist_trans.groupby('card_id').agg(aggs)
hist_trans_group.columns = new_columns
hist_trans_group.reset_index(drop=False,inplace=True)
hist_trans_group['hist_purchase_amount_diff'] =hist_trans_group['hist_purchase_amount_max']-h
hist_trans_group['hist_purchase_date_diff'] = (hist_trans_group['hist_purchase_date_max'] - h
hist_trans_group['hist_purchase_date_average'] = hist_trans_group['hist_purchase_date_diff']/
hist_trans_group['hist_purchase_date_uptonow'] = (datetime.datetime.today() - hist_trans_grou
train = train.merge(hist_trans_group,on='card_id',how='left')
test = test.merge(hist_trans_group,on='card_id',how='left')
del hist_trans_group;gc.collect()
```

```
63
```

```
train['hist_purchase_amount_sum']
```

```
0       -165.968735
1       -210.006332
2        -29.167391
3        -49.491364
4        -48.687656
            ...
19430    -50.019966
19431   -431.735321
19432    -10.638801
19433      0.348991
19434    -90.075104
Name: hist_purchase_amount_sum, Length: 19435, dtype: float32
```

```
#https://stackoverflow.com/questions/33817380/workalendar-europe-library-how-to-find-holidays
aggs = {}
for col in ['purchase_amount','month','hour','weekofyear','dayofweek','year','subsector_id','
    aggs[col] = ['nunique']
aggs['purchase_amount'] = ['sum','max','min','mean','var']
aggs['installments'] = ['sum','max','min','mean','var']
```

```
aggs['purchase_date'] = ['max','min']
aggs['month_lag'] = ['max','min','mean','var']
aggs['month_diff'] = ['mean']
aggs['weekend'] = ['sum', 'mean']
aggs['category_1'] = ['sum', 'mean']
aggs['card_id'] = ['size']

for col in ['category_2','category_3']:
    new_trans[col+'_mean'] = new_trans.groupby([col])['purchase_amount'].transform('mean')
    aggs[col+'_mean'] = ['mean']

new_columns = get_new_columns('new_hist',aggs)
hist_trans_group = new_trans.groupby('card_id').agg(aggs)

hist_trans_group.columns = new_columns
hist_trans_group.reset_index(drop=False,inplace=True)
hist_trans_group['new_hist_purchase_amount_diff']=hist_trans_group['new_hist_purchase_amount_
hist_trans_group['new_hist_purchase_date_diff'] = (hist_trans_group['new_hist_purchase_date_m
hist_trans_group['new_hist_purchase_date_average'] = hist_trans_group['new_hist_purchase_date
hist_trans_group['new_hist_purchase_date_uptonow'] = (datetime.datetime.today() - hist_trans_
train = train.merge(hist_trans_group,on='card_id',how='left')
test = test.merge(hist_trans_group,on='card_id',how='left')
del hist_trans_group;gc.collect()
```

```
22
```

```
train['new_hist_purchase_amount_sum']
```

```
0       -13.242188
1        -4.355469
2        -0.700195
3        -4.656250
4       -19.921875
           ...
19430    -2.214844
19431    -4.640625
19432    -0.566406
19433    -0.527832
19434     0.301270
Name: new_hist_purchase_amount_sum, Length: 19435, dtype: float16
```

```
del hist_trans;gc.collect()
del new_trans;gc.collect()
```

```
0
```

```
train.shape
```

```
(201917, 80)
```

```
train['outliers'] = 0
train.loc[train['target'] < -30, 'outliers'] = 1
train['outliers'].value_counts()
```

```
0    19434
1        1
Name: outliers, dtype: int64
```

```
for df in [train,test]:
    df['first_active_month'] = pd.to_datetime(df['first_active_month'], errors='coerce')
    df['dayofweek'] = df['first_active_month'].dt.dayofweek
    df['weekofyear'] = df['first_active_month'].dt.weekofyear
    df['month'] = df['first_active_month'].dt.month

    df['elapsed_time'] = (datetime.date(2018, 2, 1) - df['first_active_month'].dt.date).dt.da
    df['hist_first_buy'] = (pd.to_datetime(df['hist_purchase_date_min'])- df['first_active_mo

    #df['new_hist_first_buy'] = pd.to_datetime(df['new_hist_purchase_date_min'], errors='coer

    for f in ['hist_purchase_date_max','hist_purchase_date_min',]:

      df[f] = df[f].astype(np.int64) * 1e-9
    #df['card_id_total'] = df['new_hist_card_id_size']+df['hist_card_id_size']
    df['purchase_amount_total'] = df['new_hist_purchase_amount_sum']+df['hist_purchase_amount
```

```
for f in ['feature_1','feature_2','feature_3']:
    order_label = train.groupby([f])['outliers'].mean()
    train[f] = train[f].map(order_label)
    test[f] = test[f].map(order_label)
```

some features for the national holidays

```
# Dealing with the one nan intest.first_active_month a bit arbitrarily for now
test.loc[test['first_active_month'].isna(),'first_active_month'] =test.iloc[11577]['first_act
```

```
train['first_active_month'].dropna
test['first_active_month'].dropna
```

```
def nationalholiday(df):


  df['date'] =df['first_active_month'].dt.date


  #df = df.dropna(subset=['date'])
  #df['date'] = df['date'].astype(int)
```

```
  #df['date'] = df['date'].dt.date

    #df['date']=df['date'].isnull()
  df['day_diff1'] = df['date'].apply(lambda x: cal.get_working_days_delta(x,cal.holidays(int(

  df['day_diff2'] = df['date'].apply(lambda x: cal.get_working_days_delta(x,cal.holidays(int(
  df['day_diff3'] = df['date'].apply(lambda x: cal.get_working_days_delta(x,cal.holidays(int(
  df['day_diff4'] = df['date'].apply(lambda x: cal.get_working_days_delta(x,cal.holidays(int(
  df['day_diff5'] = df['date'].apply(lambda x: cal.get_working_days_delta(x,cal.holidays(int(
  df['day_diff6'] = df['date'].apply(lambda x: cal.get_working_days_delta(x,cal.holidays(int(
  df['day_diff7'] = df['date'].apply(lambda x: cal.get_working_days_delta(x,cal.holidays(int(
  df['day_diff8'] = df['date'].apply(lambda x: cal.get_working_days_delta(x,cal.holidays(int(
  df.drop(['date'],axis=1,inplace=True)
  return df
for df in [train, test]:
    nationalholiday(df)


train['day_diff1']

     0        107
     1          0
     2        150
     3        173
     4        214
             ...
     19430    214
     19431      0
     19432    214
     19433    128
     19434    150
     Name: day_diff1, Length: 19435, dtype: int64


gc.collect()

     145


target = train['target']
del train['target']


train.shape

     (19435, 94)


train['outliers'].describe()

     count    19435.000000
     mean         0.000051
     std          0.007173
     min          0.000000
     25%          0.000000
```

```
50%          0.000000
75%          0.000000
max          1.000000
Name: outliers, dtype: float64
```

```
train=reduce_mem_usage(train)
```

```
Mem. usage decreased to  4.11 Mb (65.5% reduction)
```

```
test=reduce_mem_usage(test)
```

```
Mem. usage decreased to 26.64 Mb (64.5% reduction)
```

```
for col in train_columns:
    for df in [train, test]:
        if df[col].dtype == "float64" :
            df[col] = df[col].fillna(df[col].mean())
```

model without outlier

```
param = {'num_leaves': 31,
         'min_data_in_leaf': 30,
         'objective':'regression',
         'max_depth': -1,
         'learning_rate': 0.01,
         "min_child_samples": 20,
         "boosting": "gbdt",
         "feature_fraction": 0.9,
         "bagging_freq": 1,
         "bagging_fraction": 0.9 ,
         "bagging_seed": 11,
         "metric": 'rmse',
         "lambda_l1": 0.1,
         "verbosity": -1,
         "nthread": 4,
         "random_state": 4590}
folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=4590)
oof = np.zeros(len(train))
predictions = np.zeros(len(test))
feature_importance_df = pd.DataFrame()

for fold_, (trn_idx, val_idx) in enumerate(folds.split(train,train['outliers'].values)):
    print("fold {}".format(fold_))
    trn_data = lgb.Dataset(train.iloc[trn_idx][train_columns], label=target.iloc[trn_idx])#,
    val_data = lgb.Dataset(train.iloc[val_idx][train_columns], label=target.iloc[val_idx])#,

    num_round = 10000
    clf = lgb.train(param, trn_data, num_round, valid_sets = [trn_data, val_data], verbose_ev
    oof[val_idx] = clf.predict(train.iloc[val_idx][train_columns], num_iteration=clf.best_ite
```

```
        oof[val_idx] = clf.predict(train.iloc[val_idx][train_columns], num_iteration=clf.best_it

        fold_importance_df = pd.DataFrame()
        fold_importance_df["Feature"] = train_columns
        fold_importance_df["importance"] = clf.feature_importance()
        fold_importance_df["fold"] = fold_ + 1
        feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)
        pickle.dump(clf, open('model.pkl','wb'))

        predictions += clf.predict(test[train_columns], num_iteration=clf.best_iteration) / folds
        pickle.dump(test[train_columns],open('test.pkl','wb'))
    np.sqrt(mean_squared_error(oof, target))
```

```
fold 0
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 2.13468        valid_1's rmse: 2.21818
[200]   training's rmse: 2.02453        valid_1's rmse: 2.16877
[300]   training's rmse: 1.95736        valid_1's rmse: 2.15481
[400]   training's rmse: 1.90597        valid_1's rmse: 2.15014
[500]   training's rmse: 1.86185        valid_1's rmse: 2.14786
[600]   training's rmse: 1.82278        valid_1's rmse: 2.14722
[700]   training's rmse: 1.78732        valid_1's rmse: 2.14661
[800]   training's rmse: 1.75391        valid_1's rmse: 2.14626
Early stopping, best iteration is:
[789]   training's rmse: 1.75752        valid_1's rmse: 2.14594
fold 1
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 2.15443        valid_1's rmse: 2.14329
[200]   training's rmse: 2.04549        valid_1's rmse: 2.0876
[300]   training's rmse: 1.97872        valid_1's rmse: 2.06643
[400]   training's rmse: 1.9263 valid_1's rmse: 2.05705
[500]   training's rmse: 1.88177        valid_1's rmse: 2.05193
[600]   training's rmse: 1.8433 valid_1's rmse: 2.04885
[700]   training's rmse: 1.80815        valid_1's rmse: 2.04868
Early stopping, best iteration is:
[665]   training's rmse: 1.81984        valid_1's rmse: 2.04805
fold 2
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 2.11506        valid_1's rmse: 2.30869
[200]   training's rmse: 2.00611        valid_1's rmse: 2.25373
[300]   training's rmse: 1.93804        valid_1's rmse: 2.23708
[400]   training's rmse: 1.88527        valid_1's rmse: 2.23
[500]   training's rmse: 1.84048        valid_1's rmse: 2.22667
[600]   training's rmse: 1.80063        valid_1's rmse: 2.22559
[700]   training's rmse: 1.76483        valid_1's rmse: 2.2257
Early stopping, best iteration is:
[673]   training's rmse: 1.77392        valid_1's rmse: 2.22543
fold 3
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 2.15912        valid_1's rmse: 2.11984
[200]   training's rmse: 2.04955        valid_1's rmse: 2.0711
[300]   training's rmse: 1.98169        valid_1's rmse: 2.0543
[400]   training's rmse: 1.92948        valid_1's rmse: 2.04733
[500]   training's rmse: 1.88621        valid_1's rmse: 2.04326
[600]   training's rmse: 1.8472 valid_1's rmse: 2.04059
```

```
[700]   training's rmse: 1.81213        valid_1's rmse: 2.03949
[800]   training's rmse: 1.7782 valid_1's rmse: 2.03867
[900]   training's rmse: 1.74636        valid_1's rmse: 2.03853
Early stopping, best iteration is:
[885]   training's rmse: 1.75072        valid_1's rmse: 2.03842
fold 4
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 2.13296        valid_1's rmse: 2.2269
[200]   training's rmse: 2.02133        valid_1's rmse: 2.1812
[300]   training's rmse: 1.95384        valid_1's rmse: 2.16729
[400]   training's rmse: 1.90064        valid_1's rmse: 2.16218
[500]   training's rmse: 1.8556 valid_1's rmse: 2.16078
[600]   training's rmse: 1.81659        valid_1's rmse: 2.15967
[700]   training's rmse: 1.78035        valid_1's rmse: 2.15934
Early stopping, best iteration is:
[646]   training's rmse: 1.79909        valid_1's rmse: 2.15901
2.124550504142991
```

```
sample=pd.read_csv('/content/drive/MyDrive/elo-merchant-category-recommendation/sample .csv')
```

```
test2 = pd.read_csv('/content/drive/MyDrive/elo-merchant-category-recommendation/test.csv')

sub_df = pd.DataFrame({"card_id":test2["card_id"].values})
sub_df["target"] = predictions
output=sub_df.to_csv("/content/drive/MyDrive/elo-merchant-category-recommendation/sample.csv'
```

```
pickle.dump(output, open('output.pkl','wb'))
```

```
model_without_outliers = pd.DataFrame({"card_id":test["card_id"].values})
model_without_outliers["target"] = predictions
```

Model with outlier

```
import lightgbm as lgb
```

```
target = train['outliers']# target as outlier
del train['outliers']
del train['target']
```

```
target
```

```
param = {'num_leaves': 31,
         'min_data_in_leaf': 30,
         'objective':'regression',
         'max_depth': -1,
         'learning_rate': 0.01,
         "min_child_samples": 20,
```

```
            "boosting": "gbdt",
            "feature_fraction": 0.9,
            "bagging_freq": 1,
            "bagging_fraction": 0.9 ,
            "bagging_seed": 11,
            "metric": 'rmse',
            "lambda_l1": 0.1,
            "verbosity": -1,
            "nthread": 4,
            "random_state": 4590}
    folds = StratifiedKFold(n_splits=5, shuffle=True, random_state=4590)
    oof = np.zeros(len(train))
    predictions = np.zeros(len(test))
    feature_importance_df = pd.DataFrame()

    for fold_, (trn_idx, val_idx) in enumerate(folds.split(train,target.values)):
        print("fold {}".format(fold_))
        trn_data = lgb.Dataset(train.iloc[trn_idx][train_columns], label=target.iloc[trn_idx])#,
        val_data = lgb.Dataset(train.iloc[val_idx][train_columns], label=target.iloc[val_idx])#,

        num_round = 10000
        clf = lgb.train(param, trn_data, num_round, valid_sets = [trn_data, val_data], verbose_ev
        oof[val_idx] = clf.predict(train.iloc[val_idx][train_columns], num_iteration=clf.best_ite

        fold_importance_df = pd.DataFrame()
        fold_importance_df["Feature"] = train_columns
        fold_importance_df["importance"] = clf.feature_importance()
        fold_importance_df["fold"] = fold_ + 1
        feature_importance_df = pd.concat([feature_importance_df, fold_importance_df], axis=0)

        predictions += clf.predict(test[train_columns], num_iteration=clf.best_iteration) / folds

    np.sqrt(mean_squared_error(oof, target))
```

```
fold 0
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 0.0984315      valid_1's rmse: 0.101132
[200]   training's rmse: 0.0959424      valid_1's rmse: 0.10058
[300]   training's rmse: 0.0942604      valid_1's rmse: 0.10043
[400]   training's rmse: 0.0931251      valid_1's rmse: 0.10042
[500]   training's rmse: 0.0923068      valid_1's rmse: 0.100391
[600]   training's rmse: 0.0916622      valid_1's rmse: 0.100365
[700]   training's rmse: 0.0910175      valid_1's rmse: 0.10035
[800]   training's rmse: 0.0904555      valid_1's rmse: 0.100335
Early stopping, best iteration is:
[797]   training's rmse: 0.0904688      valid_1's rmse: 0.100334
fold 1
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 0.0985199      valid_1's rmse: 0.100733
[200]   training's rmse: 0.0960572      valid_1's rmse: 0.100041
[300]   training's rmse: 0.0944393      valid_1's rmse: 0.0998316
[400]   training's rmse: 0.0933132      valid_1's rmse: 0.0997375
```

```
[500]   training's rmse: 0.09245        valid_1's rmse: 0.0996937
[600]   training's rmse: 0.0917665      valid_1's rmse: 0.0996833
[700]   training's rmse: 0.0911619      valid_1's rmse: 0.0996689
Early stopping, best iteration is:
[689]   training's rmse: 0.0912142      valid_1's rmse: 0.0996645
fold 2
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 0.0987877      valid_1's rmse: 0.100338
[200]   training's rmse: 0.0963708      valid_1's rmse: 0.0994179
[300]   training's rmse: 0.0947573      valid_1's rmse: 0.0992079
[400]   training's rmse: 0.0936673      valid_1's rmse: 0.0991796
Early stopping, best iteration is:
[364]   training's rmse: 0.094022       valid_1's rmse: 0.0991729
fold 3
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 0.0985401      valid_1's rmse: 0.100761
[200]   training's rmse: 0.0960786      valid_1's rmse: 0.100108
[300]   training's rmse: 0.0944458      valid_1's rmse: 0.0999225
[400]   training's rmse: 0.0933073      valid_1's rmse: 0.0998718
[500]   training's rmse: 0.0924687      valid_1's rmse: 0.0998747
Early stopping, best iteration is:
[406]   training's rmse: 0.0932505      valid_1's rmse: 0.0998649
fold 4
Training until validation scores don't improve for 100 rounds.
[100]   training's rmse: 0.0988033      valid_1's rmse: 0.100355
[200]   training's rmse: 0.0964127      valid_1's rmse: 0.0995224
[300]   training's rmse: 0.0948008      valid_1's rmse: 0.0992638
[400]   training's rmse: 0.0936348      valid_1's rmse: 0.0991431
[500]   training's rmse: 0.0927519      valid_1's rmse: 0.0991323
Early stopping, best iteration is:
[481]   training's rmse: 0.092908       valid_1's rmse: 0.0991251
0.099633238636244
```

```
### 'target' is the probability of whether an observation is an outlier
df_outlier_prob = pd.DataFrame({"card_id":test["card_id"].values})
df_outlier_prob["target"] = predictions
df_outlier_prob.head()
```

	card_id	target
0	C_ID_0ab67a22ab	0.080177
1	C_ID_130fd0cbdd	0.000697
2	C_ID_b709037bc5	0.010230
3	C_ID_d27d835a9f	0.000245
4	C_ID_2b5e3df5c2	0.002239

```
# In case missing some predictable outlier, we choose top 25000 with highest outliers likelyh
outlier_id = pd.DataFrame(df_outlier_prob.sort_values(by='target',ascending = False).head(250
```

```
sample = pd.read_csv("/content/drive/MyDrive/elo-merchant-category-recommendation/sample.csv'
```

```
most_likely_liers = sample.merge(outlier_id,how='right')
most_likely_liers.head()
```

	card_id	target
0	C_ID_ac114ef831	-20.716573
1	C_ID_cde027fde7	-20.716937
2	C_ID_a74b12dcf8	-21.321168
3	C_ID_70c457436a	-20.145637
4	C_ID_6ab591cf62	-21.469336

```
for card_id in most_likely_liers['card_id']:
    model_without_outliers.loc[model_without_outliers['card_id']==card_id,'target']\
    = most_likely_liers.loc[most_likely_liers['card_id']==card_id,'target'].values
```

```
model_without_outliers.to_csv("/content/drive/MyDrive/elo-merchant-category-recommendation/cc
```

```
model = pickle.load(open('model.pkl','rb'))
```

```
output=pickle.load(open('output.pkl','rb'))
```