Merchant Category Recommendation

Group 10: Chuchu Jin, Runlai Yang, Qi Su

***Abstract*—Personalized offers have obvious advantages in increasing transaction volume. This project mainly uses lightGBM, a boosting framework to predict user loyalty, thereby helping the company to push offers. We processed empty data and date data, considered outliers, and used bagging and boosting methods to reduce variance and bias.**

I. Background

This topic is adopted from Kaggle Competition, the original goal of the topic is to help understand the customer loyalty from the credit card information and recommend personalized promotions.
Elo has built machine learning models to understand the general aspects and preferences on analysis of all customers, but so far these features are not specified for an individual. In order to help provide the personalized promotions for customers, our goal is to come up with a model that analyzes the individual loyalty to merchants as references to the personalization.

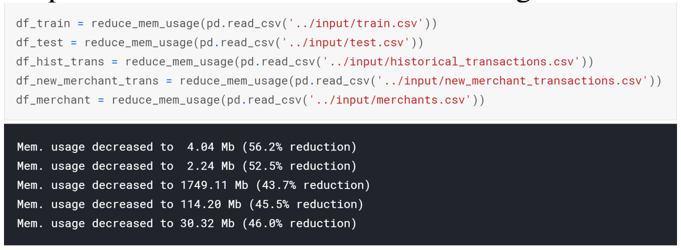
II. Data Introduction

1) The new_merchant_transactions.csv file mainly contains information about each customer's card transaction, such as card_id, merchant_id, category_# and purchase_date.
2) The merchants.csv file mainly contains merchant_id, category_#, avg_purchases, and active_month, etc.
3) The train.csv file contains first month active purchases, including card_id, feature_# (which should be found by ourselves), and target.
4) The test.csv file is also the first month active purchases, including card_id, feature_#, and then we should give a target for the result.

III. Data processing

A. Reduce memory usage

Due to extreme large scale of raw data, we use minimal data type to store the data, which could keep the most information of the original data.

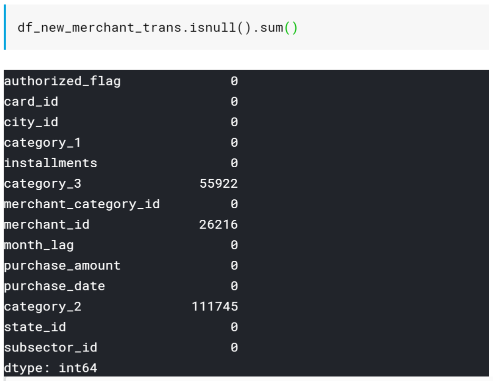


```
df_train = reduce_mem_usage(pd.read_csv('../input/train.csv'))
df_test = reduce_mem_usage(pd.read_csv('../input/test.csv'))
df_hist_trans = reduce_mem_usage(pd.read_csv('../input/historical_transactions.csv'))
df_new_merchant_trans = reduce_mem_usage(pd.read_csv('../input/new_merchant_transactions.csv'))
df_merchant = reduce_mem_usage(pd.read_csv('../input/merchants.csv'))
```

```
Mem. usage decreased to  4.04 Mb (56.2% reduction)
Mem. usage decreased to  2.24 Mb (52.5% reduction)
Mem. usage decreased to 1749.11 Mb (43.7% reduction)
Mem. usage decreased to 114.20 Mb (45.5% reduction)
Mem. usage decreased to 30.32 Mb (46.0% reduction)
```

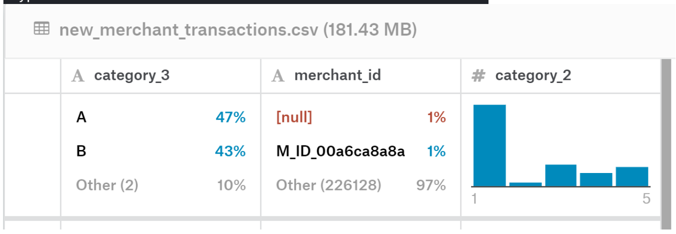
B. Handle 0 or empty data

Due to the large amount of NULL values, we filled them with most frequently occurring values.



```
df_new_merchant_trans.isnull().sum()
```

```
authorized_flag             0
card_id                     0
city_id                     0
category_1                  0
installments                0
category_3              55922
merchant_category_id        0
merchant_id             26216
month_lag                   0
purchase_amount             0
purchase_date               0
category_2             111745
state_id                    0
subsector_id                0
dtype: int64
```



new_merchant_transactions.csv (181.43 MB)

category_3		merchant_id		category_2
A	47%	[null]	1%	
B	43%	M_ID_00a6ca8a8a	1%	
Other (2)	10%	Other (226128)	97%	

C. Processing date data

The pure date data is meaningless, so we have to split the date to different parts, such as which year it is, which week in a year, which day in a week, whether it is a weekend, etc.

```
for df in [df_hist_trans,df_new_merchant_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})
    #https://www.kaggle.com/c/elo-merchant-category-recommendation/discussion/73244
    df['month_diff'] = ((datetime.datetime.today() - df['purchase_date']).dt.days)//30
    df['month_diff'] += df['month_lag']
```

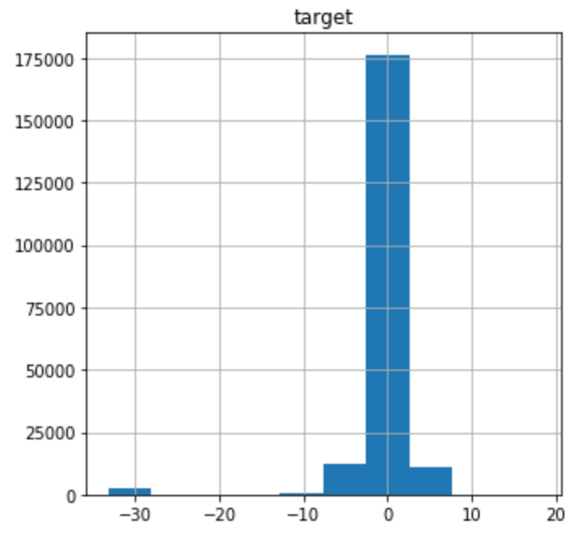
D. *Extend features*

Because we have many different tables, we need to ”join” the tables together based on the associated fields. Also, we calculate min, max, mean, variance of the numerical data to extend them.

```
for col in ['month','hour','weekofyear','dayofweek','year','subsector_id','merchant_id','mercha
nt_category_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['authorized_flag'] = ['sum', 'mean']
aggs['weekend'] = ['sum', 'mean']
aggs['category_1'] = ['sum', 'mean']
aggs['card_id'] = ['size']
```

E. *Outlier*



We can see from the picture above that most targets are located in range from -10 to 10, but there is some targets located in range around -30.

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

```
0    199710
1      2207
Name: outliers, dtype: int64
```

There is almost 1% of the data are outliers, which is already a large part of the data. We cannot give up directly, so we must find a way to deal with it.

IV. Methods

A. *LightGBM*

Many lifting tools use pre-sorted algorithms for learning decision trees. This is a simple solution, but it is not easy to optimize. LightGBM uses histogram-based algorithms to speed up training and reduce memory usage by segmenting continuous feature (attribute) values into discrete bins. Some advantages of LightGBM:

- Faster training speed and higher efficiency.
- Lower memory usage.
- Better accuracy.
- Support of parallel and GPU learning.
- Capable of handling large-scale data.

B. *Bagging and Boosting*

1) Bagging
   Based on the principle of bagging and random return sampling, a random attribute selection is introduced, and a part of features is randomly selected each time. Based on this, the optimal attributes are divided to further enhance the diversity of weak learners.
   Some advantages of bagging:
   - Parallel, no dependencies between classifiers
   - Multiple sampling to train multiple models
   - Reduce variance after fusion
2) Boosting
   Adaboost:
   - Alters the distribution of the training dataset to increase weights on sample observations that are difficult to classify.
   - The final prediction is based on a majority vote of the weak learner's predictions weighted by their individual accuracy.

   Gradient Boost:
   - The approach trains learners based upon minimizing the loss function of a strong learner (i.e., training on the residuals of the strong model) as an alternative means to focus on training upon misclassified observations.

- The contribution of each weak learner to the final prediction is based on a gradient optimization process to minimize the overall error of the strong learner.

Gradient boosting re-defines boosting as a numerical optimization problem where the objective is to minimize the loss function of the model by adding weak learners using a gradient-descent like procedure. As gradient boosting is based on minimizing a loss function.
Intuitively, gradient boosting is a stage-wise additive model that generates learners during the learning process (i.e., trees are added one at a time, and existing trees in the model are not changed). It builds a first learner to predict the observations in the training dataset, and then it calculates the loss (i.e., the value between the first learner's outcomes and the actual values). It will build a second learner that is fitted/trained on the residual error produced by the first learner to predict the loss after the first step and continue to do so until it reaches a threshold (i.e., residuals are zero). By training the second learner on the gradient of the error with respect to the loss predictions of the first learner, the second learner is being trained to correct the mistakes of the first model. This is the core of gradient boosting, and allows many simple learners to compensate for each others weaknesses to better fit the data.
Gradient boosting does not modify the sample distribution as weak learners train on the remaining residual errors of a strong learner (i.e, pseudo-residuals). By training on the residuals of the model, this is an alternative means to give more importance to misclassified observations. Intuitively, new weak learners are being added to concentrate on the areas where the existing learners are performing poorly.
The contribution of the weak learner to the ensemble is based on a gradient descent optimization process. The calculated contribution of each tree is based on minimizing the overall error of the strong learner.

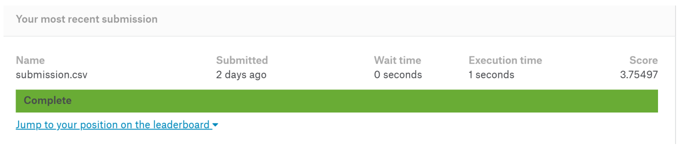
C. Handle outliers

We use three models to process outlier. The first model is trained by all the data, the second one is trained without all the outliers, and the third model is to determine whether the data is an outlier or not. Thus, we can decide which model to use depending on the third model.

V. Results

Due to the limitation of machine performance, we did not get good results. Using rmes only got a score of 3.75, while the first rank on kaggle got a score of about 3.61.

A. Our RMES score



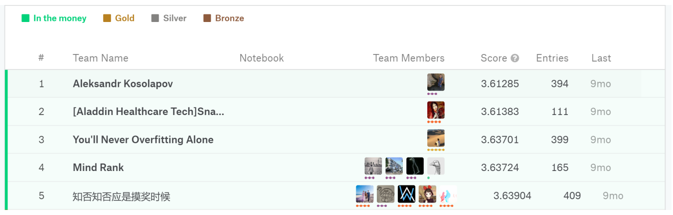
Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission.csv	2 days ago	0 seconds	1 seconds	3.75497

Complete

Jump to your position on the leaderboard

B. Other scores on Kaggle



🟩 In the money 🟧 Gold ⬜ Silver 🟫 Bronze

#	Team Name	Notebook	Team Members	Score	Entries	Last
1	Aleksandr Kosolapov			3.61285	394	9mo
2	[Aladdin Healthcare Tech]Sna...			3.61383	111	9mo
3	You'll Never Overfitting Alone			3.63701	399	9mo
4	Mind Rank			3.63724	165	9mo
5	知否知否应是摸奖时候			3.63904	409	9mo

VI. Evaluation and Improvement

In real world problem, the data set is very large. In order to get better results, we need to expand more features. Due to machine performance limitations, we did not get very good results.

1) Consider special holidays because they have a great impact on consumption.
2) For the choice of hyperparameters, we may choose a better hyperparameter if we have more time and a higher computation performance machine.
3) Try Stacking method to see if we can obtain a better result.

VII. Github Link

https://github.com/mayflywing/Machine-Learning-Elo-Merchant-Category-Recommendation

References

[1] LighGBM official documentation https://lightgbm.readthedocs.io/en/latest/

[2] DART: Dropouts meet Multiple Additive Regression Trees by K. V. Rashmi, Ran Gilad-Bachrach (7 May 2015) https://arxiv.org/abs/1505.01866

[3] Adaptive Boosting vs Gradient Boosting https://randlow.github.io/posts/machine-learning/boosting-explain/