Repeat Buyers Prediction-Challenge the Baseline*

Nian Zhonglin*
nianzhl@shanghaitech.edu.cn
(* denotes for equal contribution)

Zheng Junren*
zhengjr@shanghaitech.edu.cn
(* denotes for equal contribution)

ABSTRACT

Great benefits could be brought to merchants if potential repeated buyers are predicted successfully. In this project, we use **Light GBM** model to solve the above problem based on the long-term user behavior log accumulated by Tmall.com. We got an AUC score of 0.683952, ranking 21st (top 2%) on the Tianchi scoreboard (till 21st, June, 2018).

KEYWORDS

Machine Learning, Boosting, Bagging

ACM Reference format:
Nian Zhonglin* and Zheng Junren*. 2016. Repeat Buyers Prediction-Challenge the Baseline. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 5 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

It is important for merchants to identify who can be converted into repeated buyers. By targeting on these potential loyal customers, merchants can greatly reduce the promotion cost and enhance the return on investment (ROI). We tried several models to solve this problem, concluding that **Light GBM** has the best performance. The complete description of this problem can be found in https://tianchi.aliyun.com/getStart/information.htm?spm\=5176.11165320.5678.2.59243c3au0k7ck&raceId=231576

2 METHODS

2.1 Data Understanding & Data Preparation

The data set can be downloaded from https://tianchi.aliyun.com/getStart/information.htm?spm=5176.11165268.5678.2.73bb2a17ORD05a&raceId=231576. We use data format 2, where data fields include user_id, age_range, gender, merchant_id, label and activity_log. Here, the value of label could be {0, 1, -1, NULL}. 1 denotes user_id is a repeat buyer for merchant_id, while 0 is the opposite. -1 represents that user_id is not a new customer of the given merchant, thus out of our prediction. Our mission is to predict NULL labels, which only occurs in the testing data. We read the data from train_format2.csv and test_format2.csv, store them as training data and predict data for further usage. The section below explains how we process the data.

*Produces the permission block, and copyright information

2.2 Merge train and test

An important trick is to merge the training and predict data. Thus, we could handle the data uniformly, especially in feature engineering. Before training the model, split the training and predict data according to the label (rows with label = NULL are to be predicted.)

2.3 Drop rows with label = -1

Rows with label = -1 represents that user_id is not a new customer of the given merchant, thus out of our prediction. So we drop these rows and get a dataframe as below:

```
Data columns (total 6 columns):
user_id        522341 non-null int64
age_range      522341 non-null float64
gender         522341 non-null float64
merchant_id    522341 non-null int64
label          260864 non-null float64
activity_log   522224 non-null object
```

2.4 Handle unknown data on age and gender

User' s gender: 0 for female, 1 for male, 2 and NULL for unknown. We first replace NULL with 2, then observe the data distribution on gender.

```
df[gender].value_counts()
output:
0.0    352691
1.0    148094
2.0    21556
Name: gender, dtype: int64
```

We decide to replace gender = 2 with gender = 0 since 0 is mode and 2 is a relatively minor category.

```
df[gender].value_counts()
output:
0.0    374247
1.0    148094
Name: gender, dtype: int64
```

User' s age range: 1 for < 18; 2 for [18,24]; 3 for [25,29]; 4 for [30,34]; 5 for [35,39]; 6 for [40,49]; 7 and 8 for >= 50; 0 and NULL for unknown. We first replace NULL with 0 and replace age_range = 8 with age_range = 7 (both represent for age >= 50). Then observe the data distribution on age_range.

```
df[age_range].value_counts()
```

output:
3.0 138524
0.0 114245
4.0 102598
2.0 62385
5.0 51232
6.0 43382
7.0 9947
1.0 28
Name: age_range, dtype: int64

We decide NOT to replace age_range = 0 because it's not minor, instead, we keep it as a category.

2.5 Unfold Activity log

An activity log includes a set of interaction records between {user_id, merchant_id}, where each record is an action represented as item_id: category_id: brand_id: time_stamp: action_type. There might be several records between {user_id, merchant_id} in one row and # is used to separate two neighboring records.

An example of activity log:
293244: 1401: 2276: 1010: 2 # 917794: 1401: 2276: 1010: 2

For action_type, it is an enumerated type 0, 1, 2, 3, where 0 is for click, 1 is for add-to-cart, 2 is for purchase and 3 is for add-to-favorite. We split up activity_log and get the dataframe below:

Data columns (total 14 columns):
user_id 522341 non-null int64
age_range 522341 non-null float64
gender 522341 non-null float64
merchant_id 522341 non-null int64
label 260864 non-null float64
item_list 522341 non-null object
cat_list 522341 non-null object
brand_list 522341 non-null object
time_list 522341 non-null object
act_times 522341 non-null int64
click 522341 non-null int64
add_cart 522341 non-null int64
purchase 522341 non-null int64
add_fav 522341 non-null int64

(click/ add_cart/ purchase/ add_fav represents how many corresponding actions between {user_id, merchant_id})

2.6 Dummy Coding

Since age_range is a category variable, we decide to do dummy coding on it.

2.7 Create new features

Generally speaking, the AUC score will go up if some magic features could be excavated, so we decide to add some features as below:

(average number of actions analysis for a certain merchant: mid denotes for merchant_id)
mid_mean_click_times,
mid_mean_add_cart,
mid_mean_purchase,
mid_mean_add_fav

(customers' gender analysis for a certain merchant:)
mid_mean_gender

(customers' age analysis for a certain merchant:)
mid_mean_age_range_1.0
mid_mean_age_range_2.0,
mid_mean_age_range_3.0,
mid_mean_age_range_4.0,
mid_mean_age_range_5.0,
mid_mean_age_range_6.0,
mid_mean_age_range_7.0,
mid_mean_age_range_0.0

(time_stamp analysis for a certain merchant,
5 means May, 6 means June, etc.:)
mid_time_5,
mid_time_6,
mid_time_7,
mid_time_8,
mid_time_9,
mid_time_10,
mid_time_11

(average number of actions analysis for a certain user: uid denotes for user_id)
uid_mean_click_times,
uid_mean_add_cart,
uid_mean_purchase,
uid_mean_add_fav

(number of item/category/brand/action times for a certain user:)
uid_item_count,
uid_cat_count,
uid_brand_count,
uid_time_count

(number of item/category/brand/action times for a certain pair of merchant and user:)
uid_mid_item_count,
uid_mid_cat_count,
uid_mid_brand_count,
uid_mid_time_count

(A user's number of item/category/brand/action times on a certain merchant divides this user's total item/category/brand/action times :)
uid_mid_item_ratio,
uid_mid_cat_ratio,

uid_mid_brand_ratio,
uid_mid_time_ratio

Till now, we get a dataframe as below:

Data columns (total 62 columns):
user_id 522341 non-null int64,
gender 522341 non-null float64,
merchant_id 522341 non-null int64,
label 260864 non-null float64,
act_times 522341 non-null int64,
click 522341 non-null int64,
add_cart 522341 non-null int64,
purchase 522341 non-null int64,
add_fav 522341 non-null int64,
time_5 522341 non-null int64,
time_6 522341 non-null int64,
time_7 522341 non-null int64,
time_8 522341 non-null int64,
time_9 522341 non-null int64,
time_10 522341 non-null int64,
time_11 522341 non-null int64,
time_cov 522341 non-null float64,
age_range_0.0 522341 non-null uint8,
age_range_1.0 522341 non-null uint8,
age_range_2.0 522341 non-null uint8,
age_range_3.0 522341 non-null uint8,
age_range_4.0 522341 non-null uint8,
age_range_5.0 522341 non-null uint8,
age_range_6.0 522341 non-null uint8,
age_range_7.0 522341 non-null uint8,
mid_label_ratio 522341 non-null float64,
mid_mean_click_times 522341 non-null float64,
mid_mean_add_cart 522341 non-null float64,
mid_mean_purchase 522341 non-null float64,
mid_mean_add_fav 522341 non-null float64,
mid_mean_gender 522341 non-null float64,
mid_mean_age_range_1.0 522341 non-null float64,
mid_mean_age_range_2.0 522341 non-null float64,
mid_mean_age_range_3.0 522341 non-null float64,
mid_mean_age_range_4.0 522341 non-null float64,
mid_mean_age_range_5.0 522341 non-null float64,
mid_mean_age_range_6.0 522341 non-null float64,
mid_mean_age_range_7.0 522341 non-null float64,
mid_mean_age_range_0.0 522341 non-null float64,
mid_time_5 522341 non-null float64,
mid_time_6 522341 non-null float64,
mid_time_7 522341 non-null float64,
mid_time_8 522341 non-null float64,
mid_time_9 522341 non-null float64,
mid_time_10 522341 non-null float64,
mid_time_11 522341 non-null float64,
uid_mean_click_times 522341 non-null float64,
uid_mean_add_cart 522341 non-null float64,
uid_mean_purchase 522341 non-null float64,
uid_mean_add_fav 522341 non-null float64,
uid_item_count 522341 non-null int64,
uid_mid_item_count 522341 non-null int64,
uid_mid_item_ratio 522341 non-null float64,
uid_cat_count 522341 non-null int64,
uid_mid_cat_count 522341 non-null int64,
uid_mid_cat_ratio 522341 non-null float64,
uid_brand_count 522341 non-null int64,
uid_mid_brand_count 522341 non-null int64,
uid_mid_brand_ratio 522341 non-null float64,
uid_time_count 522341 non-null int64,
uid_mid_time_count 522341 non-null int64,
uid_mid_time_ratio 522341 non-null float64

2.8 Drop user_id and merchant_id

These two columns are meaningless while training, so we drop them.

2.9 Remove Highly Correlated Feature

We calculate the correlation matrix and drop one of two columns if the correlation coefficient is larger than 0.95 between them. As a result, click, mid_time_11, time_cov are dropped.

2.10 PCA

We keep 50 columns(Top 99.99%) in our principle components and other columns are dropped.

2.11 Model Training

After trying several models, we choose Light GBM model at last, which performs better than other boosting algorithms. We'll show you all the models we've ever tried in Experiments section. We choose Light GBM because it's memory-friendly and it requires shorter training time, but gives higher accuracy, which makes the training process easier. The figure below significantly illustrates the training time of Xgboost, Xgboost (approximate version) and Light GBM on different data sets (Higgs, Yahoo LTR, MS LTR and Expo). Y-axis denotes for training time (in seconds). Obviously, Light GBM saves us a lot of time.

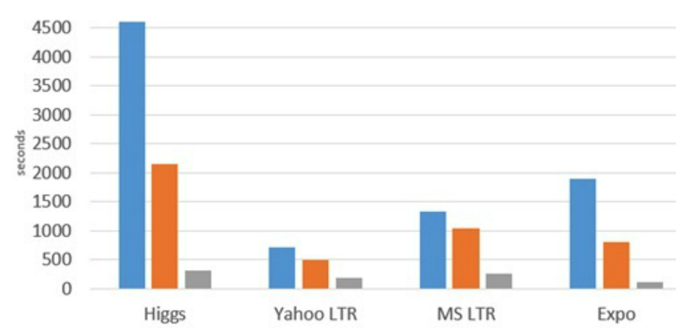

Figure 1: The comparison of different boosting algorithms based on four data sets. (Blue for Xgboost, orange for Xgboost-approx, gray for Light GBM) [1]

2.12 ROC, AUC and Confusion Matrix

Sometimes ACCURACY value on cross validation can not be used to show how our model performs perfectly, especially in label-imbalance case. For example, without considering oversampling, the majority (about 94%) of **label** in the training data is 0. We may actually achieve a high accuracy of over 90% by always predicting 0, which is very untenable.

Thus, we decide to use **confusion matrix** to summarize the performance of our classification algorithm.

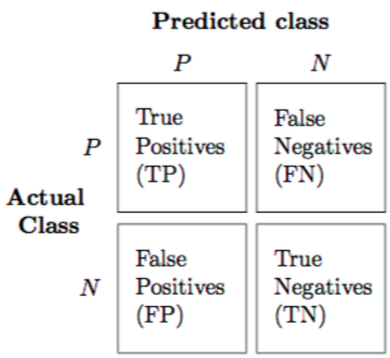

Figure 2: Confusion Matrix

The confusion matrix could give us information about **the exact type of errors (TP, FN, FP and TN)**.

From confusion matrix, we could draw **ROC** curve (ROC stands for Receiver Operating Characteristic). A point on **ROC** curve corresponds to a confusion matrix.

For each confusion matrix, we calculate two indicators, **TPR** (True positive rate) and **FPR** (False positive rate),

$$TPR = TP/(TP + FN)$$

$$FPR = FP/(FP + TN)$$

Use **FPR** as x-axis and **TPR** as y-axis to get the **ROC** curve. Although the **ROC** curve tells which classifier works better from its shape, we need a numerical value to evaluate the model's performance, which introduces **AUC** value (Area Under the **ROC** Curve). Generally speaking, the higher the **AUC** is, the better the model's performance.

2.13 Training Results

After making several tries, we find that AUC score on training data is generally higher than AUC score on 5-folder cross validation test data, which concludes to overfitting. In order to get a better score on Tianchi scoreboard, we decide to use BAGGING method to improve the generalization ability. Similar to Random Forest, we train multiple Light GBM classifiers with data bootstrapping and subsets of features. Take voting results of these classifiers to generate the final result. Finally, we get a relatively high score (0.683952 AUC score on Tianchi website) despite there is still a gap between our local CV score (0.71) and test score on Tianchi website.

	Training Score	CV Score
LIght GBM	0.775	0.7104 (+/- 0.0031)
Light GBM With Bagging	0.782	0.7122 (+/- 0.0024)

From the above table, we can say that the standard deviation (the value in the parenthesis) is obviously reduced through bagging (from +/- 0.0031 to +/- 0.0024). In other words, the generalization ability improves.

3 EXPERIMENTS

Besides the Light GBM model, we also try some other models to figure out which fits the given data set best. The models include Logistic Regression, SVM(timeout because it's difficult to train in parallel), Decision Tree, Random Forest, KNN, Stacking, Adaboost, etc. We basically tune the parameters of these models with the help of grid search. In addition, we leave out a part of training data as test data which is used for testing. The result indicates that there is nearly no difference between the CV score and the leave-out test score.

In order to deal with class imbalance situation[2], we try SMOTE algorithm and add a penalty item while training a model (class_weight = balanced). The result indicates that adding a penalty item performs better. A possible explanation is that SMOTE only generates data within the space of given data, which may not give us a better generalization ability than adding a penalty item.

The following table shows the performance of models we have ever tried.

	Accuracy on Training data	AUC on CV
LIght GBM	0.775	0.7104 (+/- 0.0031)
Light GBM With Bagging	0.782	0.7122 (+/- 0.0024)
Random Forest	0.776	0.7058 (+/- 0.0066)
Logistic Regression	0.718	0.7116 (+/- 0.0101)
StackingClassifier	0.692	0.6466 (+/- 0.0024)

Confusion Matrix:

Light GBM:
TNR: 0.69, FPR:0.31, FNR:0.29, TPR:0.71

Light GBM with Bagging:
TNR: 0.74, FPR:0.26, FNR:0.34, TPR:0.66

Random Forest:
TNR: 0.73, FPR:0.27, FNR:0.34, TPR:0.66

Logistic Regression:
TNR: 0.69, FPR:0.31, FNR:0.29, TPR:0.71

In stacking classifier, we use Logistic Regression, Random Forest and Light GBM in the first layer. We use Logistic Regression rather than some complex models in the second layer to avoid overfitting. However, it doesn't preform well.

The AUC score of different models seems quite close to each other locally, so we mainly select the final model by comparing the scores after submitting to Tianchi website.

4 ACKNOWLEDGEMENT

We sincerely appreciate it that TAs Yan Shipeng and Zhang Songyang patiently introduced feature engineering ideas and shared the trick of model selection with us. We can hardly attain such a result without their help.

REFERENCES

[1] Boosting algorithms training time compare
[2] Deal With Class Imbalance