Report of Repeat Buyer Prediction Challenge

Yiduo Gu
School of Information Science and Technology
ShanghaiTech University
guyd@shanghaitech.edu.cn

Ziyang Wang
School of Information Science and Technology
ShanghaiTech University
wangzy4@shanghaitech.edu.cn

ABSTRACT

The problem is about to predict the probability that the buyer will be converted into repeated buyer. In this project, we have used several methods to make the prediction. The score we have got on Tianchi is 0.674151.

KEYWORDS

Machine Learning, GradientBoostClassifier, Random-Forest, XGBClassifier

1 INTRODUCTION

Figure 1: Sales in Double 11

Merchants sometimes run big promotions on particular dates, in order to attract numerous new buyers. Unfortunately, many of the attracted buyers are one-time deal hunters, and these promotions may have little long-lasting impact on sales. To alleviate this problem, it is important for merchants to identify who can be converted into repeated buyers. It is well-known that in the field of online advertising, customer targeting is extremely challenging, especially for fresh buyers. However, with the long-term user behavior log accumulated by Tmall.com, we may be able to solve this problem. Using a set of merchants and their corresponding new buyers acquired during the promotion on the "Double 11" day, we have to predict which new buyers for given merchants will become loyal customers in the future. Overview of results:

- **The AUC score of GradientBoostClassifier: 0.66716715**
- **The AUC score of RandomForest: 0.656347**
- **The AUC score of XGBClassifier: 0.67415129**

2 METHODS & EXPERIMENTS

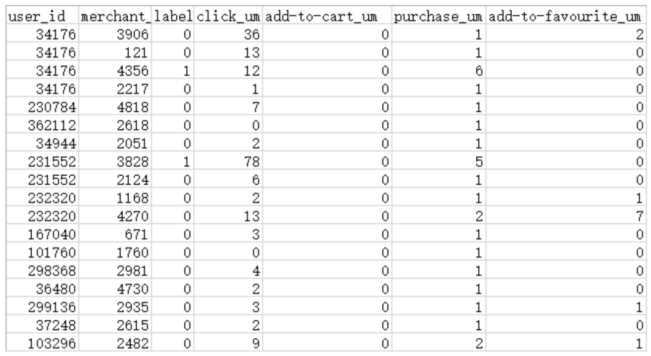
First of all, we select some features from the user_log, such as age, gender, click number, add-to-cart, purchase, add-to-favourite and so on. We use one-hot encoding method to encode the categories into binary vectors. Here is a part of our code, which is used to deal with some features:

```
onehotgender =label_binarize(np.array(
    user_dataset.gender), classes=[0, 1, 2])
onehotgender_df = pd.DataFrame(onehotgender,
    columns=['gender0','gender1','gender2'])
user_dataset['gender0'] = onehotgender_df.
    gender0
user_dataset['gender1'] = onehotgender_df.
    gender1
user_dataset['gender2'] = onehotgender_df.
    gender2

action_type_hot = label_binarize(np.array(
    user_dataset.action_type), classes=[0, 1, 2,
     3])
action_type_hot_df = pd.DataFrame(
    action_type_hot,columns=['click','addtocart'
    ,'purchase','addtofavourite'])
user_dataset['click'] = action_type_hot_df.click
```

```
user_dataset['add-to-cart'] = action_type_hot_df
    .addtocart
user_dataset['purchase'] = action_type_hot_df.
    purchase
user_dataset['add-to-favourite'] =
    action_type_hot_df.addtofavourite
```

Result:



user_id	merchant_	label	click_um	add-to-cart_um	purchase_um	add-to-favourite_um
34176	3906	0	36	0	1	2
34176	121	0	13	0	1	0
34176	4356	1	12	0	6	0
34176	2217	0	1	0	1	0
230784	4818	0	7	0	1	0
362112	2618	0	0	0	1	0
34944	2051	0	2	0	1	0
231552	3828	1	78	0	5	0
231552	2124	0	6	0	1	0
232320	1168	0	2	0	1	1
232320	4270	0	13	0	2	7
167040	671	0	3	0	1	0
101760	1760	0	0	0	1	0
298368	2981	0	4	0	1	0
36480	4730	0	2	0	1	0
299136	2935	0	3	0	1	1
37248	2615	0	2	0	1	0
103296	2482	0	9	0	2	1

At the stage of learning, there are many python libraries and analysis methods which can be used.

- **GradientBoostClassifier**
- **RandomForest**
- **XGBClassifier**

GradientBoostingClassifier:
Gradient Boosting = Gradient Descent + Boosting
The process of gradient boosting :

regression => classification => ranking

In the GradientBoostingClassifier, the experssion of the negative gradient of the loss function of the i-th sample of the t-th round is:

$$r_{ti} = -[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}]_{f(x)=f_{t-1}(x)}$$

So if we want to reduce the error as much as possible, the function should be minimum. So the output c_{tj} is:

$$c_{tj} = \arg\min_{c} \sum_{x_i \in R_{tj}} L(y_i, f_{t-1}(x_i) + c)$$

Then, we can get the final function in this round

$$h_t(x) = \sum_{j=1}^{J} c_{tj} I(x \in R_{tj})$$

By the library GradientBoostingClassifier in python, we train the model and adjust the parameter in the function. The parameters we tune in the funcion are $n\_estimators$,max_depth, $min\_samples\_split$, $max\_features$, $subsample$. The other two parameters, $learning\_rate$ and $random\_state$, are 0.1 and 10. In this method, the parameters like $n\_estimators$,max_depth, $min\_samples\_split$ have the greatest impact on the results. Therefore, modifying them are prior to adjusting the others.

Here is the final parameters:

```
X = train1.drop(['user_id','merchant_id','label'
    ],axis = 1)
y = train1['label']
from sklearn.model_selection import
    train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=.33)
gdbt = GradientBoostingClassifier(random_state
    =10,learning_rate=0.1,n_estimators=80,
    max_depth=9,min_samples_leaf=100,
    min_samples_split=1000,max_features=19,
    subsample=0.9)
gdbt.fit(X_train,y_train)
```

RandomForest:
The main idea of randomforest is simple. We should build the a forest randomly. There are many decision trees in the forest. Each tree has no connection to others. After building a forest, when a new sample input enter the forest, all of trees will classify the sample and figure out which type it is. The type which is choosen by the trees the most will be the output of randomforest.
If there is a train set with n samples, and we iterate B times, then the output should be

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(x')$$

In the library of python, the RandomForest has parameters such as $n\_estimators$,max_depth, etc. As the same way as we did to GradientBoostingClassifier, the parameters $n\_estimators$,max_depth, $min\_samples\_split$ should be tuned a head of others, since they affect the output greatly.

Here is the final parameters:

```
X = train1.drop(['user_id','merchant_id','label'
    ],axis = 1)
y = train1['label']
from sklearn.model_selection import
    train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=.33)
gdbt = RandomForestClassifier(n_estimators=70,
    max_depth=13,min_samples_split=100,
    min_samples_leaf=30,max_features=13)
gdbt.fit(X_train,y_train)
```

XGBClassifier:
XGBClassifier is more likely to be "GradientBoostingClassifier pro". It is an optimized version of the Gradient Boosting algorithm.
The difference between XGBClassifier and GradientBoostingClassifier is that XGBClassifier do the second-order Taylor expansion on the cost function.

$$g_i = \partial_{\hat{y}^{(t-1)}} L(y_i, \hat{y}^{(t-1)})$$
$$h_i = \partial^2_{\hat{y}^{(t-1)}} L(y_i, \hat{y}^{(t-1)})$$

For a given data set with n examples and m features. $D = (x_i, y_i)(|D| = n, x_i \in R^m, y_i \in R)$, a tree ensemble model uses K additive functions to predict the output.

$$\hat{y_i} = \sum_{k=1}^{K} f_k(x_i), f_k \in A,$$

where $A = \{f(x) = W_{q(x)}\}(q : R^m \to T, w \in R^T)$ is the space of regression trees. Here q represents the structure of each tree that maps an example to rhe corresponding leaf index.Unlike decision trees, each regression tree contains a continuous score on each of the leaf, we use w_i to represent score on i^{th} leaf.
Second-order approximation can be used to quickly optimize the objective in the general setting:

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^{n}[g_i w_q(x_i) + \frac{1}{2} h_i w^2_{q(x_i)}] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2}(\sum_{i \in I_j} h_j + \lambda) w_j^2] + \gamma T \end{aligned}$$

The weight w_j^* of leaf j:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_j + \lambda}$$

Then define:

$$G_j = \sum_{i \in I_j} g_i$$
$$H_j = \sum_{i \in I_j} h_j$$

Then the object function can be written as

$$Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

At the end of one step of the iteration, add $f_t(x)$ to the model

$$y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$$

The usage of ϵ is to aviod over-fitting.
The parameters of XGBClassifier are *n_estimators*,*max_depth*, *min_child_weight*, *gamma*,*colsample_bytree*, *subsample*, *reg_alpha*.
The steps in tunning parameters:

- max_depth and min_child_weigh
- gamma
- subsample and colsample_bytree
- reg_alpha

Here is the final parameters:

```
X = train1.drop(['user_id','merchant_id','label'
    ],axis = 1)
y = train1['label']
from sklearn.model_selection import
    train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=.33)
gdbt = RandomForestClassifier(n_estimators=70,
    max_depth=13,min_samples_split=100,
    min_samples_leaf=30,max_features=13)
gdbt.fit(X_train,y_train)
```

3 CONCLUSION

In the experiments, we use AUC to check the model whether it can predict the result well.
AUC is the area under an ROC curve.The Receiver Operating Characteristics (ROC) curve for a binary classification problem plots the true positive rate as a function of the false positive rate.The points of the curve are obtained by sweeping the classification threshold θ from the most positive classification value to the most negative. For a fully random classification, the ROC curve is a straight line connecting the origin to (1, 1). Any improvement over random classification results in an ROC curve at least partially above this straight line.
To learn the set of functions used in the model, we minimize the following regularized objective.

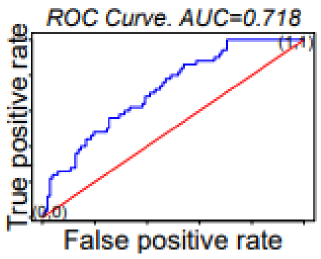


Figure 2: An example of ROC curve

$$True\ positive\ rate = \frac{correctly\ classified\ positive}{total\ positive}$$

$$False\ positive\ rate = \frac{incorrectly\ classified\ negative}{total\ negative}$$

Load the data:

```
train1=pd.read_csv('train+.csv')
test1=pd.read_csv('test+.csv')
```

GradientBoostingClassifier:

```
X = train1.drop(['user_id','merchant_id','label'
    ],axis = 1)
y = train1['label']
from sklearn.model_selection import
    train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=.33)
gdbt = GradientBoostingClassifier(random_state
    =10,learning_rate=0.1,n_estimators=80,
    max_depth=9,min_samples_leaf=100,
    min_samples_split=1000,max_features=19,
    subsample=0.9)
gdbt.fit(X_train,y_train)
y_pred = gdbt.predict(X_test)
y_predprob = gdbt.predict_proba(X_test)[:,1]
print ("AUC Score (Train): %f" % metrics.
    roc_AUC_score(y_test, y_predprob))
```

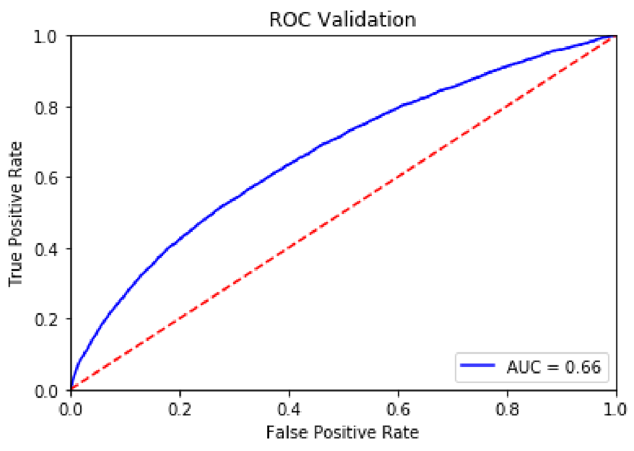


Figure 3: ROC curve of GradientBoostingClassifier

RandomForest:

```
X = train1.drop(['user_id','merchant_id','label'
    ],axis = 1)
y = train1['label']
from sklearn.model_selection import
    train_test_split
X_train, X_test, y_train, y_test =
    train_test_split( X, y, test_size=.33)
gdbt = RandomForestClassifier(n_estimators=70,
    max_depth=13,min_samples_split=100,
    min_samples_leaf=30,max_features=13)
gdbt.fit(X_train,y_train)
y_pred = gdbt.predict(X_test)
y_predprob = gdbt.predict_proba(X_test)[:,1]
print ("AUC Score (Train): %f" % metrics.
    roc_AUC_score(y_test, y_predprob))
```

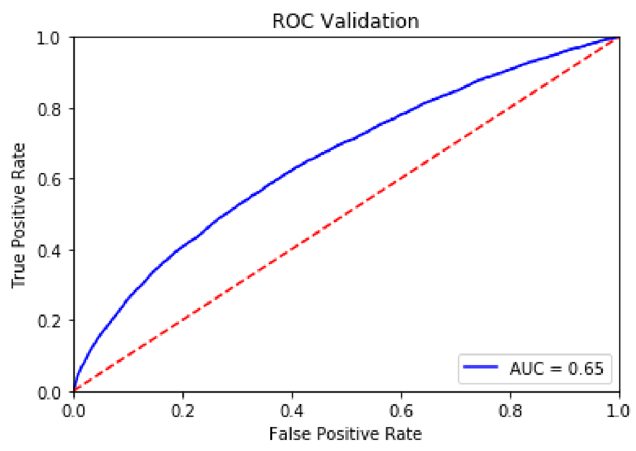


Figure 4: ROC curve of RandomForestClassifier

XGBClassifier:

```
X = train1.drop(['user_id','merchant_id','label'
    ],axis = 1)
y = train1['label']
from sklearn.model_selection import
    train_test_split
X_train, X_test, y_train, y_test =
    train_test_split(X, y, test_size=.33,
    random_state=47)
xgb =XGBClassifier(learning_rate=0.1,
    n_estimators=280,max_depth=5,
    min_child_weight=3,gamma=0.17,
    colsample_bytree=0.6,subsample=0.8,reg_alpha
    =1)
xgb.fit(X_train,y_train)
y_predprob = xgb.predict_proba(X_test)[:,1]
print ("AUC Score (Train): %f" % metrics.
    roc_AUC_score(y_test, y_predprob))
```

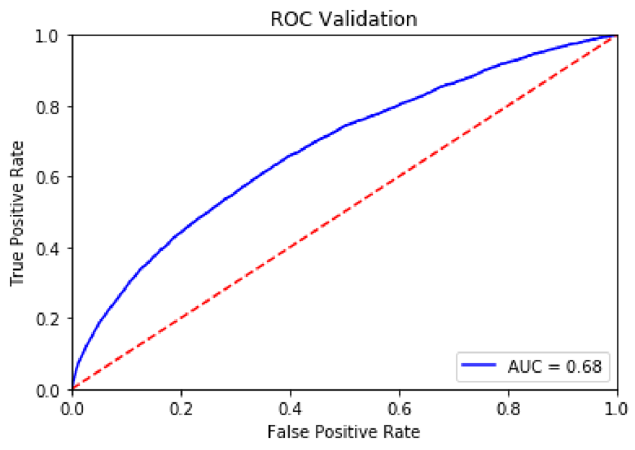


Figure 5: ROC curve of XGBClassifier

4 REFERENCE

- Leo Breiman.*RANDOM FORESTS*,University of California Berkeley, 2001.
- Gerard Biau.*Analysis of a Random Forests Mode*,France.
- Corinna Cortes and Mehryar Mohri.*AUC Optimization vs. Error Rate Minimization*,USA.
- Tianqi Chen and Carlos Guestrin.*XGBoost: A Scalable Tree Boosting System*,USA.
- Si Si, Huan Zhang, S. Sathiya Keerthi.*Gradient Boosted Decision Trees for High Dimensional Sparse Output*,USA.
- Jerome H. Friedman.*Greedy Function Approximation: A Gradient Boosting Machine*,USA,1999.

5 WORKLOAD

- **Ziyang Wang: 50%**
- **Yiduo Gu: 50%**