TalkingData: AdTracking Fraud Detection Challenge
Part2 Implementation

Lixuan Mao (lm769), Ang Li (al2386)

Abstract

After the baseline implementation, a powerful framework named LightGBM was introduced to further increase the AUC score. Apart from using the framework, a lot of efforts were put into doing Feature Engineering to derive many more useful features from existing features. Combining new model and derivative features, much higher AUC scores were achieved and some analysis were produced based on these results. [1]

1 Introduction: LightGBM

LightGBM(Ke et al., 2017) is a gradient boosting framework that uses learning algorithms. It is designed to be distributed and efficient with the following advantages:

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

By leveraging LightGBM, one can easily train several types of tree-based learning algorithms on huge datasets in a efficient way, which is perfectly suitable for the Talkingdata challenge.

In LightGBM, we specifically explored two novel tree-based models: GBDT(Gradient Boosting Decision Tree) and DART(Dropouts meet Multiple Additive Regression Trees).

GBDT(Gradient Boosting Decision Tree) is one of the baseline algorithms introduced in last part of the project. The reason why we are still interested in it is that it is a really basic and versatile classification model. Besides, by leveraging the power of LightGBM, GBDT can have higher accuracy and cost us less time to train.

DART(Dropouts meet Multiple Additive Regression Trees) is a novel model that tries to introduce the idea of dropout into **Multiple Additive Regression Trees (MART)** so that during the iteration of each tree, it will not try to fit the residual of the previous tree, but first sample a subset of the existing trees to ensemble a new model, then try to fit the residual of this new model. In this innovating way, it would make the contribution of the later generated trees larger so as to prevent over-specialization.

In this part, we will do some experiments to compare the performance of these two models.

2 Feature Engineering

2.1 GroupBy-Generated Features

The first major type of manual features generated is *GroupBy-Generated Features*. This type of features is generated by first grouping different entries that have some same basic features like 'ip', 'app', 'channel', etc., then calculating some statistics [2], such as mean, variance, unique value count, etc., of selected features, like 'minute', 'app', 'ip', etc., within each group. Finally, we used left-join on those group-by features to connect derivative features back to data entries. (Rashmi and Gilad-Bachrach, 2015) For example, in one manual features, we used three basic features 'ip','app' and 'channel' to group the dataset and calculated

[1] Implementation codes can be found here.

[2] var: variance;
mean: mean value;
count: compute count of group, excluding missing values;
nunique: return Series with number of distinct observations over requested axis;
cumcount: number each item in each group from 0 to the length of that group - 1

the variance of 'day' within each group. After that, using left-join on 'ip','app' and 'channel' can connect the new feature values, called 'ip_app_channel-var-day', back to proper data entries.

Table 1 lists all the *GroupBy-Generated Features* generated in this implementation.

In order to validate the usefulness of these manual features, we computed 5-fold cross-validation scores with one million samples using LightGBM to see how much each feature improved the model or not. The performance improvement of each feature can be seen in Figure 1 [3].

2.2 Confidence Rate Features

From the observation of the dataset, we saw that certain ips, apps, channels or certain combinations of them can lead to higher probability of application download, which is shown by higher frequency of 'is_attributed'.

Then we thought of using conditional probability to present such underline relationship so that we can have some more useful features. The general format of such conditional features can be expressed by

$$P(is\_attributed|Feature_1, Feature_2, ...) \quad (1)$$

However, there was a concern that if a given feature-combination has so few positive 'is_attributed', then the statistical significance would be far from trustworthy. Therefore, one approach adopted here was to weighting the conditional features by confidence rates:

$$\text{conf}_{\text{is\_attributed}} = \frac{\log(\text{entries}_{\text{features}})}{\log(100000)} \quad (2)$$

where the denominator item $\log(100000)$ is a scaler such that if a given feature-combination has 1000 entries, then it gets a confidence rate weight of 0.5; if it has 100 views, then it only gets a weight of 0.333, etc..

Table 2 lists all the *Confidence Rate Features* generated in this implementation.

In order to find out how useful such *Confidence Rate Features* were, a 5-fold cross-validation experiment over one million randomly picked samples using LightGBM was conducted to see how much each feature improved the model or not. The performance improvement of each newly generated feature can be seen in Figure 2.

3 Experiment Setup

After the work of Feature Engineering, we now have 40 features to use for training classification models of LightGBM. However, there are several types of boosting models to choose as well as a number of hyperparameters like learning_rare, num_leaves, max_depth, etc. to tune.

In order to find out how each hyperparameter will affect the performance of the result model, a series of experiments under different parameter settings were conducted. Table 3 displays all these parameter settings.

The training set was 2 millions randomly picked entries, and 10% of it was used to do validation.

4 Result & Analysis

In the experiments, **AUC Score** over validation set was used to evaluate the performance of the model. In order to see the how different hyperparameters(Regularization Coefficient, Maximum Depth, Minimum Data in Leaf) would affect the performance, we conducted a series of experiments using control variate method, and the plot the results into two sets of graphs, one for **GBDT** and one for **DART**. The plots were organized in this way: we fixed two of the three hyperparameters and plot the left one against AUC Score.

The graphs can be seen in Table 4.

Observing from the results, one can see that AUC scores of these models were all higher than baseline models (4% 6%), which shows the power of our works.

From the graphs, one can see that neither Regularization Coefficient(lamda) nor Minimum Data in Leaf(min_child_samples) has a significant influence over the performance of both types of models.

However, Maximum Depth(max_depth) does have an influence over the performance. One can see that too large Maximum Depth would lead to the decrease of AUC Scores, which is probably because large Maximum Depth tends to make the model over-specified resulting in the decrease the generalization ability of the models.

Additionally, one can see that DART models generally perform slightly better than GBDT models.

[3]The first box is the baseline, which is obtained by only using basic features, which are ['ip','app','device', 'os','channel','day','hour','minute', 'second'], to train.

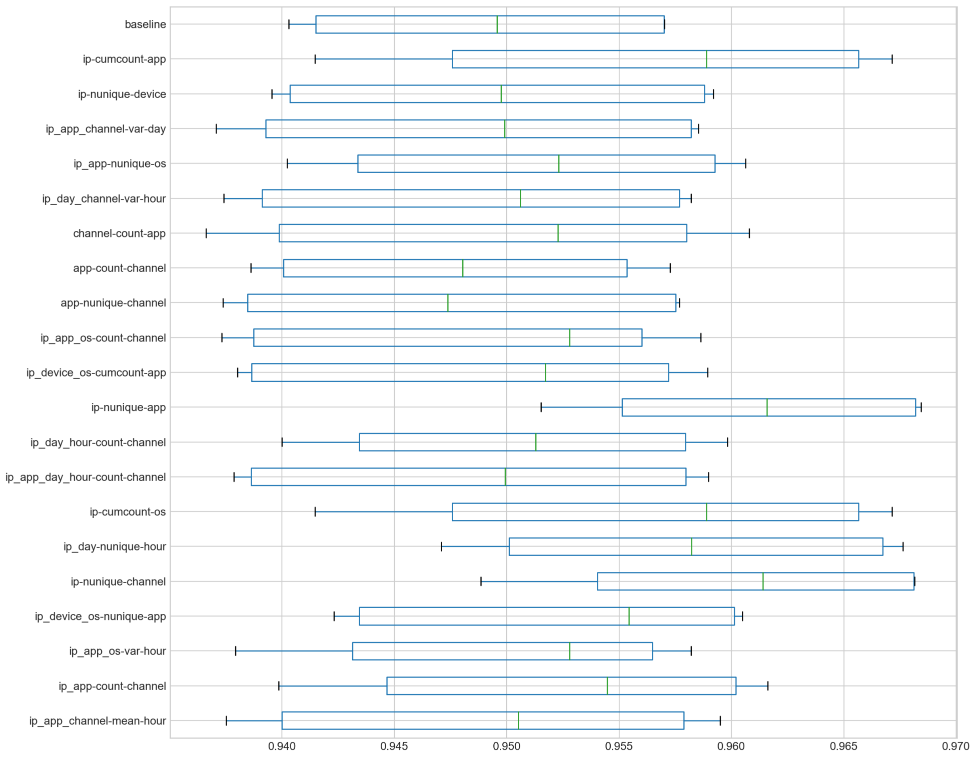


Figure 1: Group-by Features Performance Gain

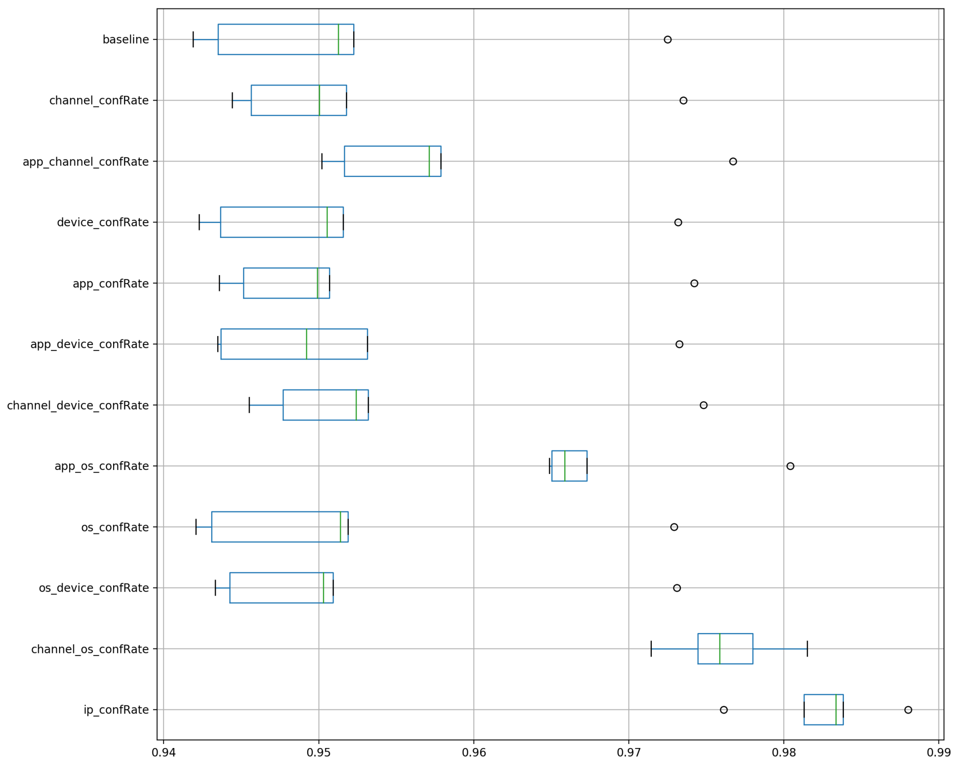


Figure 2: Confidence Rate Features Performance Gain

References

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, Curran Associates, Inc., pages 3146–3154. http://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf.

K.V. Rashmi and Ran Gilad-Bachrach. 2015.

Group-by Features	Selected Feature	Calculated Statistics
ip, app, channel	day	var
ip, app, os	hour	var
ip, day, channel	hour	var
ip, day, hour	channel	count
ip, app	channel	count
ip, app, os	channel	count
ip, app, day, hour	channel	count
ip, app, channel	hour	mean
app	channel	count
channel	app	count
ip	channel	nunique
ip	app	nunique
ip, day	hour	nunique
ip, app	os	nunique
ip	device	nunique
app	channel	nunique
ip, device, os	app	nunique
ip, device, os	app	cumcount
ip	app	cumcount
ip	os	cumcount

Table 1: GroupBy-Generated Features

Condition Features
ip
app
device
os
channel
app, channel
app, os
app, device
channel, os
channel, device
os, device

Table 2: Confidence Rate Features

Model	Regularization Coefficient	Maximum Depth	Minimum Data in Leaf
GBDT	[0.01, 0.05, 0.1, 0.5]	[2, 4, 8, no_limit]	[20, 40, 60, 100]
DART	[0.01, 0.05, 0.1, 0.5]	[2, 4, 8, no_limit]	[20, 40, 60, 100]

Table 3: Experiment Parameter Setting

Dart: Dropouts meet multiple additive regression trees. https://www.microsoft.com/en-us/research/publication/dart-dropouts-meet-multiple-additive-regression-trees/.

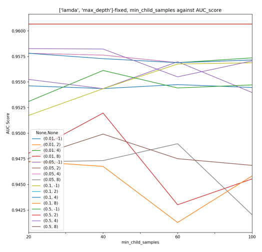
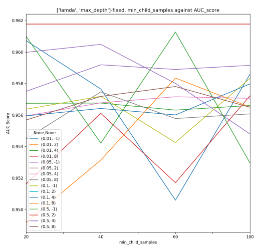
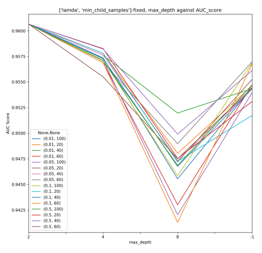
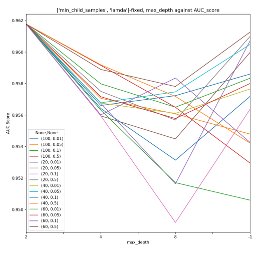
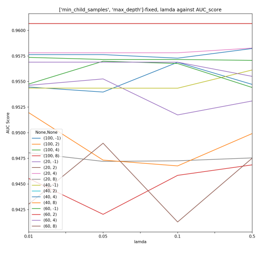
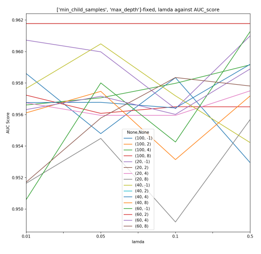
	GBDT	DART
(lamda, max_depth) fixed, min_child_samples v.s. AUC		
(lamda, min_child_samples) fixed, max_depth v.s AUC		
(max_depth, min_child_samples) fixed, lamda v.s. AUC		

Table 4: How Hyperparameters Affect Performance of Models