TalkingData AdTracking Fraud Detection Challenge: Kaggle

Pavle Mihajlovic
University of Waterloo
pavle.p.mihajlovic@gmail.com
pmihajlo@edu.uwaterloo.ca

Abstract

This report summarizes a classification analysis in the context of a Kaggle competition. The main purpose of this paper is to investigate simple feature extraction methods from high cardinality data to achieve close to state of the art performance. The best performing model performed at 0.9777 AUC-ROC which was only 0.0066 AUC-ROC away from the top performing model.

1 Introduction

The risk of fraud exists everywhere, but companies that offer advertisements online are especially at risk for click fraud. Click fraud occurs online in a pay-per-click scheme where owners of websites that post ads are paid a small amount every time a visitor to their website clicks on an ad. Malicious entities exist that are willing to employ computer scripts and/or low wage workers to intentionally generate money for the owners of the websites and costs for the advertisers. This results in misleading click data and wasted money for advertisers.

TalkingData is China's largest independent big data service platform and covers over 70% of active mobile devices in China, generating 3 billion clicks every day of which many are sources of potential click fraud. App developers pay money to advertisers to advertise their specific app in which some potential customers who click on the ad download the app and some do not. However some hostile actors exist who are willing to repeatedly click on the ads and never download the app, constituting in some cases click fraud or maybe a highly hesitant customer.

Currently TalkingData handles probable click fraud cases for app developers by measuring the journey of a user's click across their portfolio and flagging IPs which produce many clicks but never download apps.

Now they have turned to the Kaggle community to build an algorithm that predicts if a user will download an app after clicking a mobile app ad. (https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection).

2 Data and Problem

In this competition the objective is to predict whether a user will download an app after clicking a mobile app ad.

TalkingData has provided a huge dataset gathered over 4 days featuring over 203 million rows. Specifically the training set contains almost 185 million labelled samples and the test set contains almost 19 million unlabelled samples.

The training data features 8 data fields.

- ip: ip address of click (unordered categorical)
- app: app id for marketing (unordered categorical)
- device: device type id of user mobile phone (eg: iphone 7) (unordered categorical)
- os: os version id of user mobile phone (unordered categorical)

- channel: channel id of mobile ad publisher (unordered categorical)
- click_time: timestamp of click (UTC)
- attributed_time: if user downloaded the app, this is the time of the app download
- is_attributed: target to predict, indicating the app was downloaded

Note: ip, app, device, os, channel are all encoded (identity hidden). Test data is similar, with attributed_time and is_attributed removed, and click_id is an added data field which is a reference for making predictions.

This competition is a binary classification problem and the method of evaluation is AUC ROC. We are attempting to categorize into 2 categories:

- class 0: app was not downloaded after mobile app ad click
- class 1: app was downloaded after mobile app ad click

It is also impervious to note that the leaderboard is calculated with only 18% of the test data. Therefore it is more than likely that high performing models are overfitting the small test set (eg: over tuning parameters) and that intuitive and reasonable feature creation should hypothetically perform just as well computationally intensive methods.

3 Exploratory Data Analysis

There are a couple confounding factors which make this dataset hard to work with. Firstly the training set is over 7 GB large which means a plethora of sampling and feature extraction procedures should be undertaken. Additionally 5/7 data fields that we can use for prediction are categorical and feature extremely high cardinality thus one-hot-encoding those variates is an inappropriate way to deal with them. Class imbalance is another factor with only 0.247 % of the classes being positive.

For visualizations only 3 sets of the dataset are focused on due to computational memory constraints.

- The entire test set
- A large sample of class 0 in the train set
- The entire class 1 in the train set

The entire test set is small enough to completely fit into memory and knowing the characteristics of the set we are predicting on is paramount to performing well. With this in mind we wish to uncover distributional differences among the test and train sets. A large sample is chosen for the negative class in the train set (class = 0), because the set is highly imbalanced and we cannot fit all of the data in at once. The entire positive class is very small, making up only 456 thousand rows, which can be dealt with in memory.

All the categorical variables are encoded and feature no ordered characteristics. However the categorical features, especially ip, features much too many categories to be reasonably one-hot-encoded within memory.

Click_time is the only data field which features any temporal characteristic. It features the year, month, day, hour, minute and second of when the ad was clicked. Unfortunately the data was collected over a 4 day period which implies that only hour, minute and second will be of use. Note: the first and last day of the training set are incomplete. It's important to note that there exists a imbalance in the test set vs the train set with regards to the hour of the click. Certain hours make up almost all of the observations (+99%), some hours make up a couple hundred observations ($< 1\%$) and some hours have no observations whatsoever. This in direct contrast to our train set which has plenty of observations in all hours of the day with only natural decreases occurring during night.

Overall the exploratory data analysis of the variates does not reveal anything out of the ordinary when comparing the test and training sets.

3.1 Feature Creation

This subsection will cover simple but effective and intuitive methods to extract features from high cardinality data.

- Temporal Characteristics: hour, minute and second
- Frequency Characteristics: how often does some variate appear in the class we would like to predict?
- Aggregation Characteristics: how can we use aggregations to create better predictions?

Temporal: Since the data spans only 4 days extracting inference from anything larger than hour might be a waste of time. A method of adding temporal characteristics to the dataset is to try to optimize for time to next click. Most of the clicks tend to come from specific ip addresses or devices. Therefore we can create variables measuring how long until the next click comes from some combination of the categorical variates. For example: how long until ip-app-device performs their next click.

Frequency: We can reveal which categories and combinations of categories are more likely to result in click fraud. Here the idea is to create a bayesian probability for each click based on some arbitrary combination of columns being in class 1 (app was downloaded). However interpreting the new variate as a bayesian probability naively is prone to overfitting the train data since some categories will appear so little that there could be an over representation. Thus we suggest a confidence weight method of aggregating the bayesian probability so as to limit overfitting.

The simple method of calculating the frequency feature, which is highly prone to overfitting:

$$P(\ is\_attributed \mid category\ )$$

However if the above bayesian probability is re-weighted based on the number of observations in the set of categories then we should avoid overfitting to specific edge cases and idiosyncracies within the data.

Method to re-weight frequency features:

$$weight = \frac{log(\ count\ of\ category\ )}{log(100000)}$$

The value of 100000 is seemingly arbitrarily chosen, such that if a category has 1000 views, it gets a confidence weight of 60% and if it only has 100 views then only a confidence weight of 40%.

Thus our new bayesian probability is:

$$min(1\ , weight) * P(\ is\_attributed \mid category\ )$$

Aggregation: In this method we test single, double and triple interaction features. For example we create a column where for some single,double,triple mixture of distinct categorical variables where a aggregation is determined. The aggregations which we use are counts, averages and variances.

4 Methods

We start by applying a couple of very simple models. A set of training data was randomly picked with a test-train split of 0.9. Since XGBoost was the best performing model on the base variables we decide to use this as our final model as it should be more fruitful to iteratively improve on the best performing base model by introducing more intricate variates and parameter tuning.

Model	Validation
Logistic Regression	0.70
RandomForest	0.91
XGBoost	0.93

4.1 XGBoost: Why?

Xgboost otherwise known as extreme gradient boosting wins lots of kaggle competitions. It's relatively simple to understand and highly powerful. A wealth of information exists for this type of model in the form of kernels and

on the internet in general. It is one of the premier algorithms used for datascience on kaggle.

4.2 XGBoost: An Introduction

XGBoost is a supervised machine learning method where we have labelled training data and target variables to predict. Different problems call for different metrics to evaluate on. We have an objective function for which we measure the performance of a model. This objective contains a training loss and a regularization term. The training loss evaluates how our model performed and the regularization controls complexity.

XGBoost is a tree ensemble method that follows CART (classification and regression trees). A simple explanation of cart follows. Imagine you create a graph like binary tree structure and at each node you want to make a binary evaluation (to move down an edge). You start at the parent node and you make a binary decision. This binary decision is either regression based (eg: is your age over 24?) or classification based (eg: are you male?). So at the parent node you answer yes or no and go down that specific edge slowly making your way to the end of the tree structure. And once the end is reached a prediction score is assigned to each leaf. This is one tree. The idea of doing this many times over, thus creating many trees is called tree ensembles.

The idea of using this model is to be able to thoroughly test the features that have been created in a computationally intensive manner. Within the CART framework one can easily see how each of the temporal, aggregation and frequency characteristics fit in this framework and how subsequently they can be tested for whether they are effective features.

5 Results

5 sets are used to summarize the results. A point to notice is that differing sizes of training sets were used to gather the results. This was a needed sacrifice as our better performing models featured much more variables and thus fitting more data in memory was impossible. Thus an implicit trade off exists between using a larger training set compared to more explanatory variables.

- Set 1: Selected Count Characteristics only, 30 million observations
- Set 2: Frequency Characteristics only, 20 million observations
- Set 3: Temporal + selected count characteristics, 20 million observations
- Set 4: Count + Frequency Characteristics, 10 million observations
- Set 5: All features, 10 million observations

Model	LB
Set 1: Count	0.9693
Set 2: Frequency	0.9701
Set 3: Temporal + Count	0.9686
Set 4: Count + Frequency	0.9724
Set 5: All Features	0.9767

5.1 Discussion

Due to the nature of rare event classification tasks all of the sets performed reasonably well. Notably the final set #5, had the best score on the leader board which should be expected, as it aggregated all of the best features from the other models. In depth kernels where computationally intensive methods were used to choose hyperparameters. After that parameters were generally chosen ad-hoc based on AUC-ROC score.

6 Conclusion

In this report a series of methods were used to document how to carefully extract useful features from categorical data that featured high cardinality. The focus was less on hyperparameter tuning and more on meaningful feature creation. Through this competition we have displayed that one can reach state of the art performance using salient features and limited computing resources.

7 References

Template for report:
http://emnlp2014.org/templates/acl2014.pdf
Confidence weight method for feature creation:
https://www.kaggle.com/nanomathias/feature-engineering-importance-testing
Indepth EDA - Entire Set:
https://www.kaggle.com/gopisaran/indepth-eda-entire-talkingdata-dataset
Introduction to xgboost
http://xgboost.readthedocs.io/en/latest/model.html
Kaggle Master Explains Gradient Boosting:
http://blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/