Ad Tracking Fraud Detection Challenge
Shilpa Rajbhandari
December 4, 2018

1. Introduction

In pay-per-click online advertising, the amount of money paid to website owners posting the ads depends on the number of clicks on the ads on the website. This brings about a major risk of fraud for online advertisers. A person or an automated script/program can mimic a legitimate user and click on the ads thus driving costs up by simply clicking on the ad at a large scale. This type of fraud is known as click fraud.

With 1 billion smart mobile devices in active user each month, China is the largest mobile user in the world. As such, there is a tremendous volume of click fraud that happens in China. TalkingData, China's largest independent big data service platform, handles close to 3 billion clicks per day out of which 90% is likely to be fraudulent. While TalkingData has built an IP and device blacklist to prevent click fraud; it is looking to develop a solution to prevent click frauds. One way to do this is by building an algorithm that predicts if the user will download an app after clicking in the mobile apps advertisement.
Pay-per-click model of advertisement has introduced and increased risks of click fraud. Click fraud can happen when the publisher of the website themselves or by the means of a botnet of computers fraudulently click on the ads in order to generate revenue. Click fraud can also happen when competitors of an advertiser click on the ads with the motive of depleting the ad budget of the competitor. This means that click fraud presents a huge financial risk for online advertisers as their advertisement dollars are going to waste. According to a study performed in 2013 by Incapsula, over 60% of internet traffic is non-human. As such, it is important to find a solution to effectively detect instances of click fraud and block such clicks. To research this topic further, I will be analyzing data from Kaggle competition.
(https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data)

2. Data Background and Pre-processing

Elaborating on the dataset mentioned above, the data describe 200 million clicks over 4 days. Data contains a total of seven attributes: ip (address of click), app (app id for marketing), device(device type id of user mobile phone), os (os version id of user mobile phone), channel(channel id of mobile ad publisher), click time (timestamp of click), attributed time(if user download the app for after clicking an ad, this is the time of the app download), is attributed(the target that is to predicted, indicating the app was downloaded).

I began the analysis with a few data pre-processing steps, to ensure data was clean and structured properly and would obtain accurate results. First, I checked whether there were no null or missing values in data, and when that was successful, I made sure our variables were encoded properly and began to separate and assign a target and predictor variables. Next, I

randomly chose 90% of our data for training and 10% for testing, indicating a random state of 99, so our results are reproducible and stratified our data by the class labels in our Is_atrributed target variable so it structurally remains consistent. The dataset is very large, so I read 10 million rows, skipping 250 for the preprocessing so that the dataset is random and would predict well. I used 2 million rows training because of the computation power.

Target variable in this analysis is attribution which is a binary classification, with the remaining five features assigned as our predictor variables. Then, I count the unique values per feature in the predictor variable. Among them, ip has the most unique values. After this, I just looked at the exploratory analysis, distribution for value count of ip, app, device, os, channel. Looking at the target value distribution, we can see that only 0.19% has the attribute 1 which is what we are trying to predict. To see what the most frequent hour of time for app download, I split the time into day, hour, minute and sec and did the count. In a time, a span of 24 hours, 16 (4 pm) is the maximum time for app download, the second is 23 (11 pm). Furthermore, processing has been done for features (attribution category, aggregation, next click) to calculate confidence value, predictor variable having a higher frequency, variance in predictor variable, analyze how popular is the app or channel, average click on an ad by a distinct user. All the variables are encoded for privacy reasons, which means that it will be lower the feature engineering process.

3. Description of Algorithm

For the modeling, I am using XGBoost (Extreme Gradient Boosting) for the modeling. This is an ensemble method, which has high flexibility, handles missing values and involves multiple trees to make a decision. It uses regularized model validation to control over-fitting giving the best performance. XGBoost improves models on iteratively and the error are improved in the training data.

For the other model, I am using Random Forest, it is an ensemble learning, function by building a congregation of decision forest. It is a robust model and can be used both in regression and classification. This model is very good for overfitting; thus, I choose this model for prediction.

4. Results

To ensure I obtain the most accurate results possible, I will complete a robust analysis, applying two robust classification models to our data and selecting the model with the highest predictive accuracy. In the following pages, I will take you through our evaluation process for both models.

4.1 XGBoost

The first model I use was XG-boost algorithm. I set the param and pass the training and validation and train the model into the training set then predict it into target variable is_attributed. The accuracy increases in each iteration. I got an accuracy of 0.96163 for training and for validation 0.942163.

For the model, I have split train set into training and validation set, 90 per training and 10 per validation, I have created matrices from training and validation data. I pass our dataset to training our model with 100 boosting stages. Further, I have created a submission file (xgb_sub_dub_2mill.csv) for the model. I have got validation accuracy as 95.34% for XGBoost. From the test dataset, we have predicted the target variable. I have calculated the confusion matrix and ROC error Metrics.

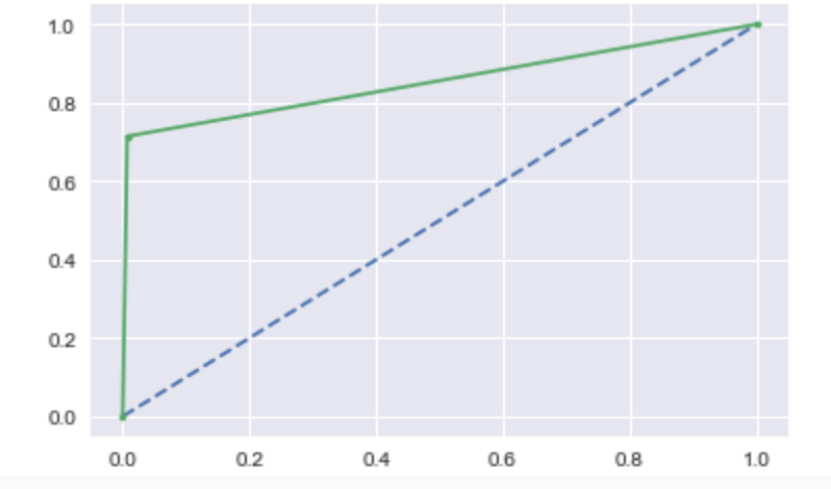

Figure 1: ROC curve for XGBoost

The ROC shows that the usefulness of a true positive rate against false positive rate, here in the figure above shows out curve shows a high overall accuracy of the test.

```
array([[99139,   700],
       [   46,   115]])
```

The confusion matrix from XGBoost shows,

Accuracy: Overall, how often is the classifier correct? 99.254%
Misclassification Rate: Overall, how often is it wrong? 0.746%
True Positive Rate: When it's actually yes, how often does it predict yes? 0.115%
False Positive Rate: When it's actually no, how often does it predict yes? 0.70%
True Negative Rate: When it's actually no, how often does it predict no? 99.29%%

4.2 Random Forest

The second model I use was a Random Forest Classifier. This classifier is relatively robust, so it is not sensitive to multicollinearity, outliers, or susceptible to overfitting. Taking these

assumptions about the algorithm into consideration, there was not much data preparation to do before training and testing the model. When I fit our model using all of our features, I got an accuracy of 0.99808.

Random forest classifiers nature is supposed to mitigate overfitting. This is one of the key reasons I use random forests instead of decision trees because they estimate a large number of decision trees and then output the mode of the results, decreasing the likelihood of overfitting. However, with may attempts to dissect what issue may be occurring, we still yet to have concluded.

5. Summary and Conclusion

Random Forest works best for the model because it is not affected by multicollinearity, it has low variance, so not susceptible to overfitting like decision trees. We can add more features to the training and testing dataset and increase our prediction. The random forest grows trees in parallel and it reduces variance iteratively whereas XGBoost after each run reduces bias and also algorithm is based on weak learners meaning high bias and low variance. The highest time for the click time in a day is 16, 17, 22 and 23. The click fraud detectors can use this time frame actively to prevent the fraud. For this competition, we have encoded dataset, but the Ad tracking company can have a real dataset, build a model to track the ip, device, channel and could predict the click fraud. Since the dataset is really big to work on a local computer we need higher computation power.

6. Reference

https://www.kaggle.com/tunguz/xgboost-starter
https://www.kaggle.com/anokas/talkingdata-adtracking-ed