PUBH 7475: Final Project Report

TalkingData AdTracking Fraud Detection

Thao Nguyen, Chunni Zhao

May 6, 2018

1 Introduction

With the development of the Internet, click fraud has increasingly becoming one of the biggest threats in the cyberworld. More and more companies are now using online advertisement as a main medium of putting their brand names out to customers, and website owners benefit from clicks make to these ads on their websites. Businesses are willing to invest in generating genuine clicks from users, which represent the users' interest and possible opportunities for these companies to create and deliver value. However, in reality, not all clicks correctly represent the clickers' interest in the product or service [1]. The click fraud phenomenon is becoming more and more widespread, with 'click farms' generating artificial clicks using human or technology to make businesses pay for their ads. This is particularly distressing for small companies, which are forced to accept fraudulent clicks as a part of the cost in their marketing budgets [1].

One of the biggest markets that are targeted by online fraudsters is the mobile market. With over 1 billion smart mobile devices in active use every month, China is the largest mobile market in the world and therefore suffers from huge volumes of fraudulent traffic. Hence, efficient fraud detection techniques are in urgent need.

1.1 Data Description

This dataset was taken from the "TalkingData AdTracking Fraud Detection Challenge" on Kaggle. The host company, TalkingData, is China's largest independent big data service platform. It covers over 70% of active mobile devices nationwide and handles 3 billion clicks per day, most of which are potentially fraudulent. For this challenge, our goal was to build an algorithm that predicts whether a user will download an application after clicking a mobile advertisement.

The dataset that TalkingData had provided covers approximately 200 million clicks. The whole dataset, which was ordered by timestamps, was split into the train set, whose first click began on 2017-11-06, and the test set, whose first click began on 2017-11-10. The train set covers all the clicks during a 3-day period, and the test set covers all the clicks on the

next day.
8 features are included in train data set (Table 1), namely `ip`, `app`, `device`, `os`, `channel`, `is_attributed`, `click_time` and `attributed_time`.
The test set contains only 7 features, with the exception of `is_attributed`, which is the binary response variable that needs to be predicted, and `attributed_time`. The test set also contains an ID column to identify each observation.
Figure 1 shows the number of unique values in our train data set. We can see that `ip`, `app`, `device`, `os` and `channel` are actually categorical variables encoded as integers.
This dataset is highly imbalanced, with 99.75% of the observations belonging to the negative class and only 0.25% of the observations in the positive class (Figure 2).

1.2 Feature Explanation

The feature `app` represents the app ID of a particular application that the user is looking at. Regarding the relationship between `app` and `channel`, there can be more than one application in one channel, and one application can be put in several channels. However, the applications inside one specific channel are fixed.
The feature `ip` contains different IP addresses. However, we thought that it was not appropriate to treat each IP as a unique value. Firstly, IP addresses are easily generated, and we would not be able to recognize a real IP from a fake one. Moreover, Figure 3 shows that a single IP can be associated with different devices on different operating systems, implying that the IP address could have been representative of a network address. This was a motivation for us to treat the aggregated feature `ip_device_os` as one single user.
Inspecting the `attributed_time` feature, we found most of the observations to be missing. However, this was due to the fact that most of the clicks did not convert into downloads (i.e. `is_attributed` equals 0). This feature is only available when a click was converted into a download, which is also why it was missing in the test data set. We plotted the behaviors of `click_time` and `attributed_time` in Figure 4 and found that they had the same patterns. Therefore, it seemed reasonable to drop the `attributed_time` feature and only extract information from the `click_time` feature. As the behaviors of the converted and non-converted clicks were possibly different, we took a look at the conversion rate, which is the proportion of download counts over the total click counts.
Figure 5 shows a seasonal behavior in click time on a daily basis, so it was reasonable for us to use `hour` and `wday` (day of the week) as new features in data analysis. Since our data only covers a 4-day span, the year, month, or week were all the same for all observations, so we did not use this information in prediction, and dropped the `click_time` feature after necessary information had been extracted.

1.3 Data Pre-processing

Our data set was highly imbalanced, as described in the previous section. In classification problems, most classifiers like logistic regression and decision tree work best when the class distribution of the response variable in the dataset is balanced. However, in real world problems, like fraud detection, the datasets are often highly imbalanced and it would be difficult to derive a meaningful and good predictive model, due to the lack of information

of the minority class [2]. There are two popular solutions to deal with imbalanced data: oversampling or undersampling. Considering the nature of this dataset (high number of observations, which would require a lot of computing power), we decided to go with the undersampling method, which randomly discarded majority samples (negative observations in `is_attributed`) to achieve equal distribution with the minority class (positive observations in `is_attributed`). We ended up with nearly 1 million observations as our new balanced train set.
However, this meant that we were potentially losing useful information in our original train dataset. Therefore, we also performed our modeling methods on two other subsets, including the first 10 or 50 million observations in the given train set. Even if these subsets were still imbalanced, we were hoping to extract some useful information, while still keeping the computation capabilities in check.
We also found that, for some operating systems, for example os19, the count of non-converted clicks was much larger compared to that of converted ones (Figure 6), which made us suspect that these clicks associated with os19 could be fraudulent. Therefore, we chose to aggregate our data on count of os to represent this latent information. Similarly, we created other aggregated features on the count of other original features (Figure 7-9).

1.4 Feature Engineering

To obtain the latent information of the conversion rate, we aggregated the training data on different groups of features. We tried several combinations of two-way and three-way interactions. For example, feature `ip_os_count` means that we grouped the data by `ip` and `os` and counted the number of clicks. We paid close attention to the features aggregated on `hour`, as we thought this might be an important feature, and those that were aggregated on `hour` proved to be significant in our models. After trying all possible combinations, our final predictive model included 21 features, and the correlation matrix among these features was illustrated in Figure 10.

2 Methods

2.1 Model Selection

For this binary classification problem, several methods were considered, namely Logistic Regression, Random Forest, and Gradient Boosting (XGBoost and LightGBM). We evaluated our models by the area under the ROC curve (AUC). According to Kaggle rules, the AUC scores on the public leaderboard were calculated using only a subset of the test data set.

2.1.1 Logistic Regression and Random Forest

We first trained our data using logistic regression (with LASSO regularization), and random forest, using the balanced training data set of 1 million observations. Logistic regression gave an AUC of 0.58, and random forest gave an AUC of 0.89. Both of these methods were very time and memory consuming, so we did not consider them for larger training sets.

2.1.2 Gradient Boosting Decision Tree

Gradient boosting decision tree (GBDT) is a state-of-the-art machine learning algorithm that has been used widely in recent years, especially in Kaggle competitions. Gradient boosting methods have proved their efficiency and accuracy over common ensemble techniques like random forest [3].
One of the most popular systems is XGBoost, which is an open source scalable tree boosting system that has had big impacts on several machine learning and data mining challenges. Some of XGBoost's advantages include its fast learning speed (using parallel and distributed computing), lower memory requirement and the ability to handle sparse data [4].
However, one drawback of XGBoost, or other conventional implementations of GBDT in general, is that it has to scan all instances of every feature to estimate the information gain of all possible split points before deciding the best split among all features [5]. This can be very time consuming while dealing with big data, in terms of both the number of instances and the number of features that we have. LightGBM solves this problem by using Gradient-based One-side Sampling (GOSS) to keep all instances with large gradients, which would contribute more to the information gain, and randomly drop instances with small gradients, while still keeping the same data distribution. GOSS has been proved to achieve a good balance between reducing the number of data instances, which makes LightGBM more time efficient than conventional GBDT methods, while still keeping the accuracy for learned decision trees. LightGBM also uses Exclusive Feature Bundling (EFB) technique, which deals with the sparse feature space problem [5].

2.2 Model Optimization

Given the efficiency and high accuracy of GBDT methods, we applied XGBoost and LightGBM to two training subsets, one with the first 10 million observations, and one with the first 50 million observations of the original train dataset. Both methods have very similar parameters that can be used in hyper-parameter tuning in model optimization. We considered a variety of parameters in the tuning process, but paid the most attention to `max_depth`, which controls the maximum depth of a tree, and `num_leaves`, which controls the number of leaves in one tree. This is particularly important in LightGBM, as this method splits trees leaf-wise, whereas XGBoost splits trees depth-wise. The leaf-wise algorithm can converge faster, but is more prone to over-fitting, as a higher depth would mean that the model will learn relations that are specific to a particular sample. Controlling `max_depth` would prevent us from growing very deep trees, thus preventing overfitting. In theory, `num_leaves` would have to be less than or equal to $2^{\texttt{max\_depth}}$, so we fixed `num_leaves` at $2^{\texttt{max\_depth}} - 1$ [6].
Another important parameter that we considered was `scale_pos_weight`, which controls the weight of positive class in our binary classification task. As our data is highly imbalanced, a heavy weight was assigned to the positive class to balance out the model.

3 Results

Logistic regression and random forest were not considered further in model evaluation, as they gave low AUC scores and were very time and memory consuming. We applied GBDT

methods on the 3 training subsets. Even with optimization, XGBoost only gave an AUC of 0.9 at best, which was not as competitive as LightGBM, which gave AUC scores of 0.95, 0.96, and 0.97, respectively, for 3 training subsets. Thus, we decided to use our optimized LightGBM model using 50 million observations in prediction.

4 Discussion

As seen in Figure 11, the newly created features were correlated with one another, and some of the original features were correlated as well (`device` and `os`). This explains why logistic regression was not a good model choice, even with LASSO regularization. Even after removing correlated features, the model was not competitive. Tree-based models solve this high multicollinearity problem by constructing trees in a greedy manner and using all information from all the newly created features. Among the GBDT methods in consideration, LightGBM proved to be a more preferable method, both in time, memory efficiency and accuracy. XGBoost could be competitive, but as we were dealing with a very large data set, LightGBM was much faster, while still producing the same accuracy.

We also learned that our AUC score increased as we used more observations in training. We tried to deal with the imbalanced data problem in our training dataset by creating a balanced undersampled train dataset. However, this left us with very little information, as most of the majority samples were discarded. Thus, using more samples and adjusting for the weight of the minority class in the GBDT methods resulted in better accuracys. This was another advantage of GBDT that was not possible in methods like logistic regression.

Various other methods have been considered for future work due to lack of time and computing resources. One direction is to try CatBoost, which was proven to be very efficient in handling categorical features by one-hot-encoding. CatBoost may behave better than both LightGBM and XGBoost if we are dealing with a lot of categorical variables [7]. We also wanted to try stacking weaker models. Since stacking typically yields better performance than a single model, it might be worth trying if computing resources are sufficient.

References

[1] Nir Kshetri. (2010). *The Economics of Click Fraud.*. IEEE Security and Privacy, Vol. 8, No. 3, pp. 45-53.

[2] B.W. Yap, K.A. Rani, H.A.A. Rahman, S. Fong, Z. Khairudin and N.N. Abdullah. (2014). *An Application of Oversampling, Undersampling, Bagging and Boosting in Handling Imbalanced Datasets.* Proceedings of the First International Conference on Advanced Data and Information Engineering, Lecture Notes in Electrical Engineering 285.

[3] A. Natekin and A. Knoll. (2013). *Gradient boosting machines, a tutorial.* Front. Neurorobot 7:21.

[4] T. Chen and C. Guestrin. (2016). *XGBoost: A Scalable Tree Boosting System.* KDD'16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 785-794.

[5] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye and T. Liu. (2017). *LightGBM: A Highly Efficient Gradient Boosting Decision Tree.* 31st Conference on Neural Information Processing Systems.

[6] Microsoft: Parameters Tuning. (2017).
`https://lightgbm.readthedocs.io/en/latest/Parameters-Tuning.html`

[7] A.V. Dorogush, V. Ershov and A. Gulin. (2017). *CatBoost: gradient boosting with categorical features support.*

Appendices

A Tables

Feature name	Feature Meaning
`ip`	IP address of click
`app`	App ID for marketing
`device`	Device type of user mobile phone
`os`	OS version of user mobile phone
`channel`	Channel ID of mobile ad publisher
`click_time`	Timestamp of click (UTC)
`attributed_time`	Timestamp when an app is downloaded
`is_attributed`	1 if the app is download, 0 otherwise

Table 1. Description of all features

B Graphs

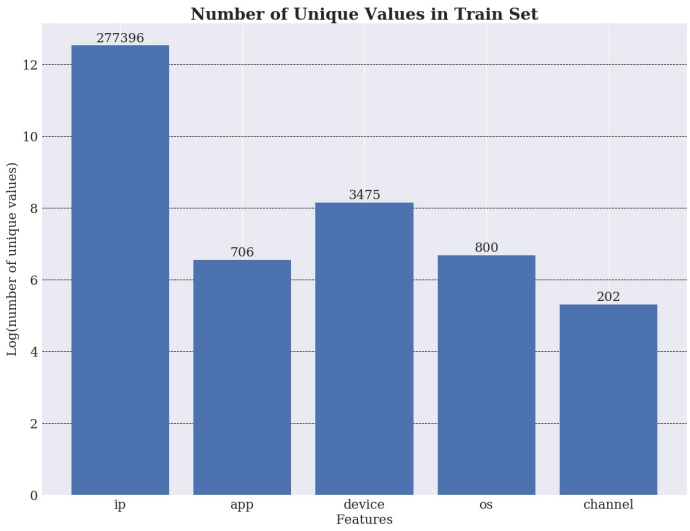

Figure 1. Number of unique values in train set.

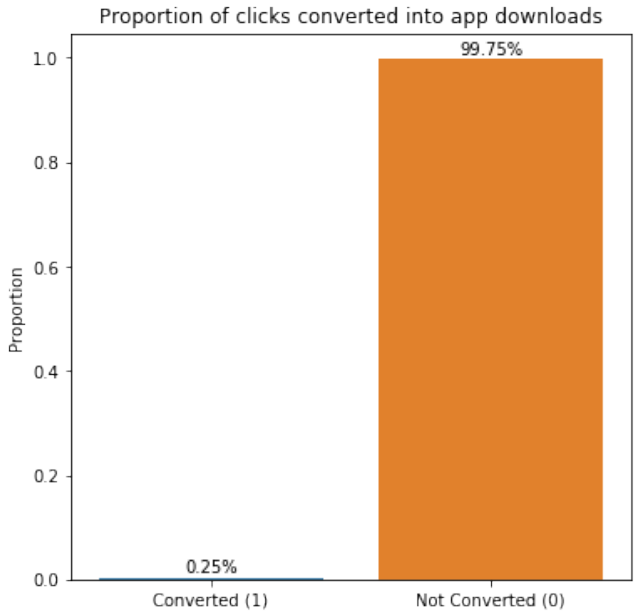

Figure 2. Proportion of clicks converted into app downloads.

ip	device	os
6	1	23
6	1	19
6	1	22
6	1	16
6	14	24
6	545	24

Figure 3. Sample distribution of `ip`, `device`, and `os` for `ip` $= 6$.

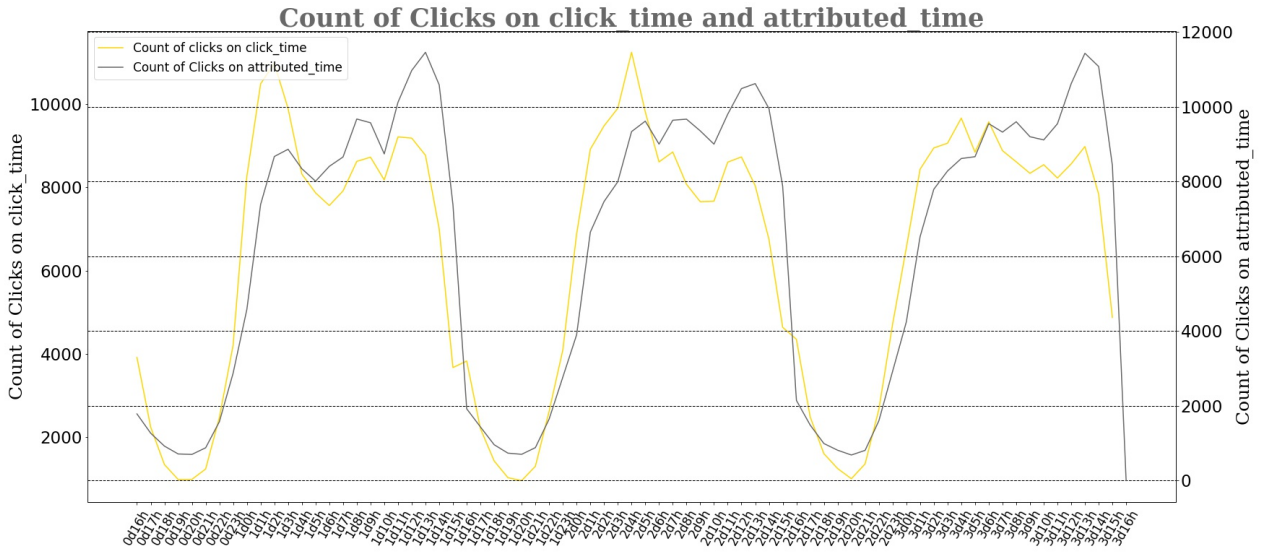

Figure 4. Time patterns of `click_time` and `attributed_time`.

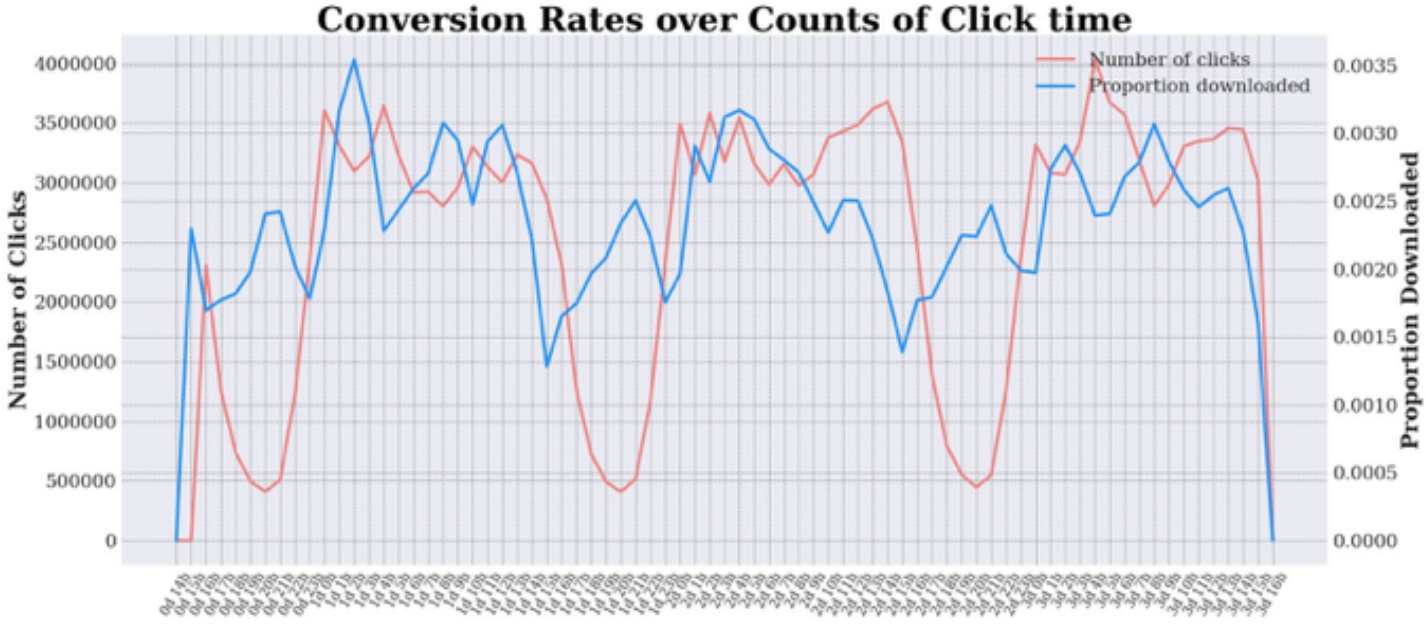

Figure 5. Conversion rates over counts of `click_time`.

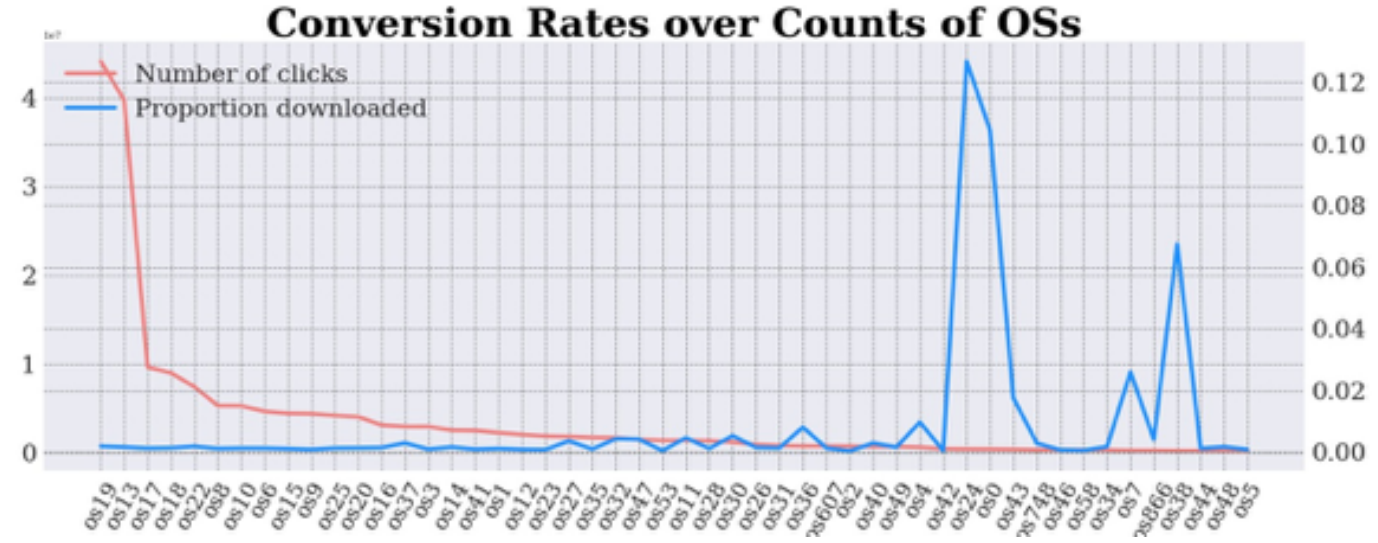

Figure 6. Conversion rates over counts of `os`.

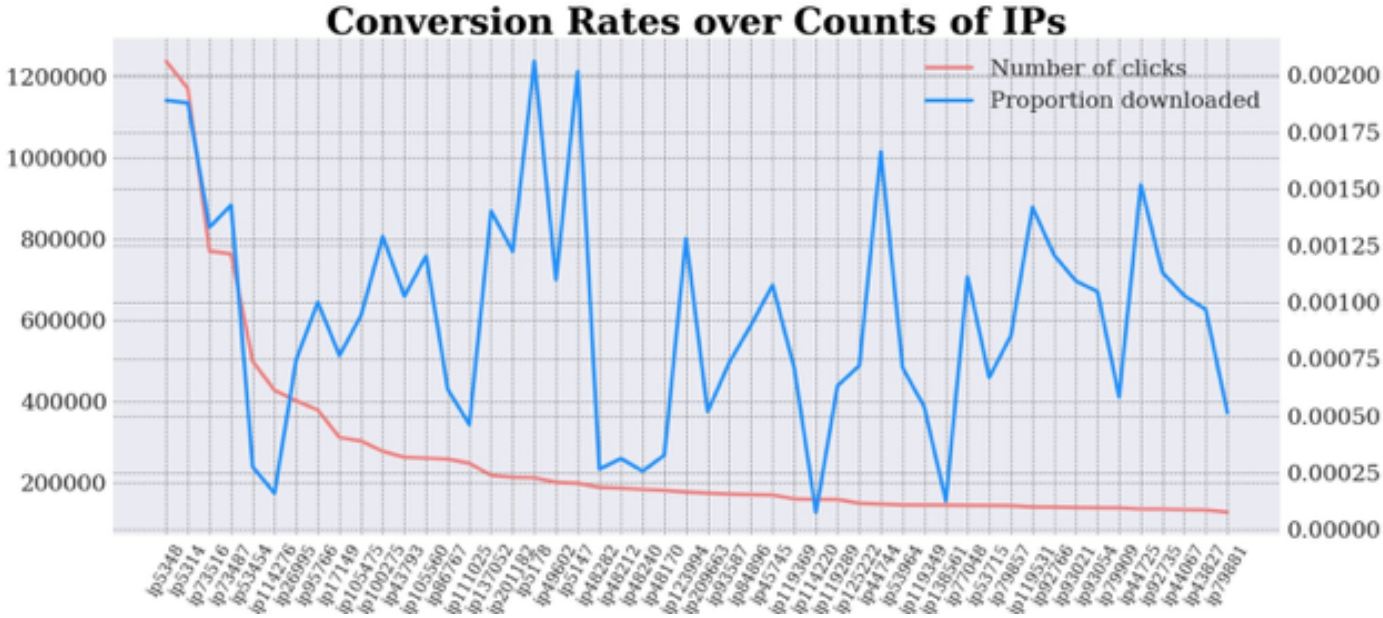

Figure 7. Conversion rates over counts of `ip`.

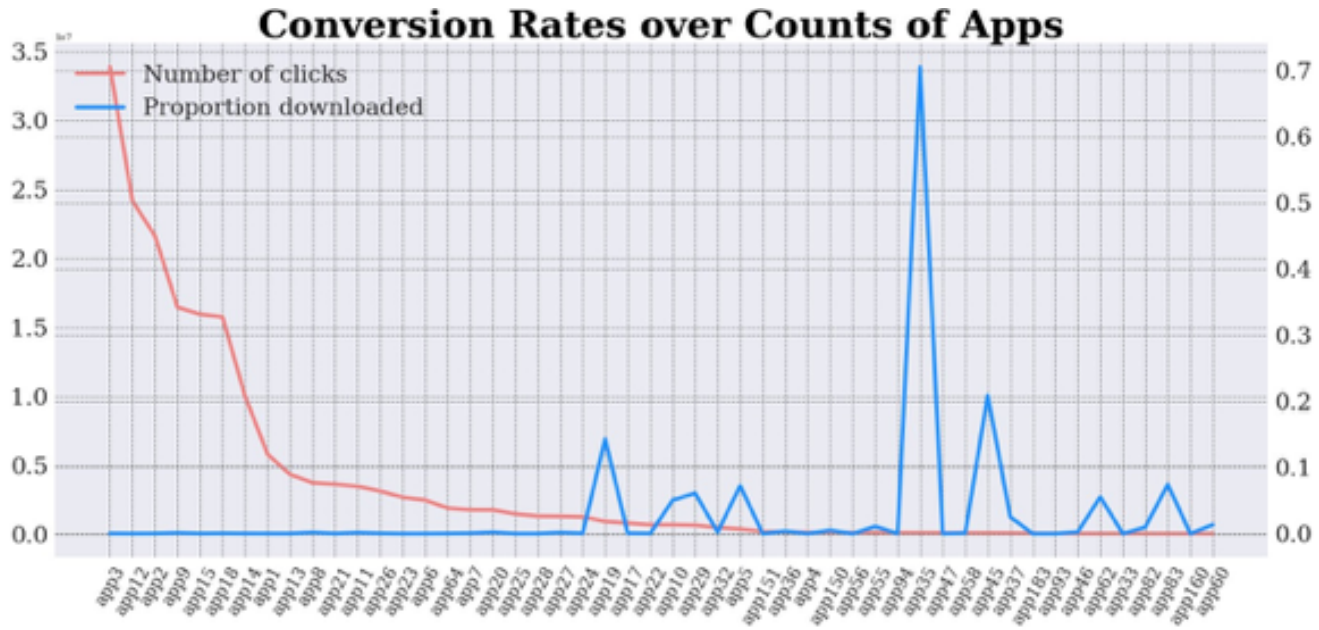

Figure 8. Conversion rates over counts of `app`.

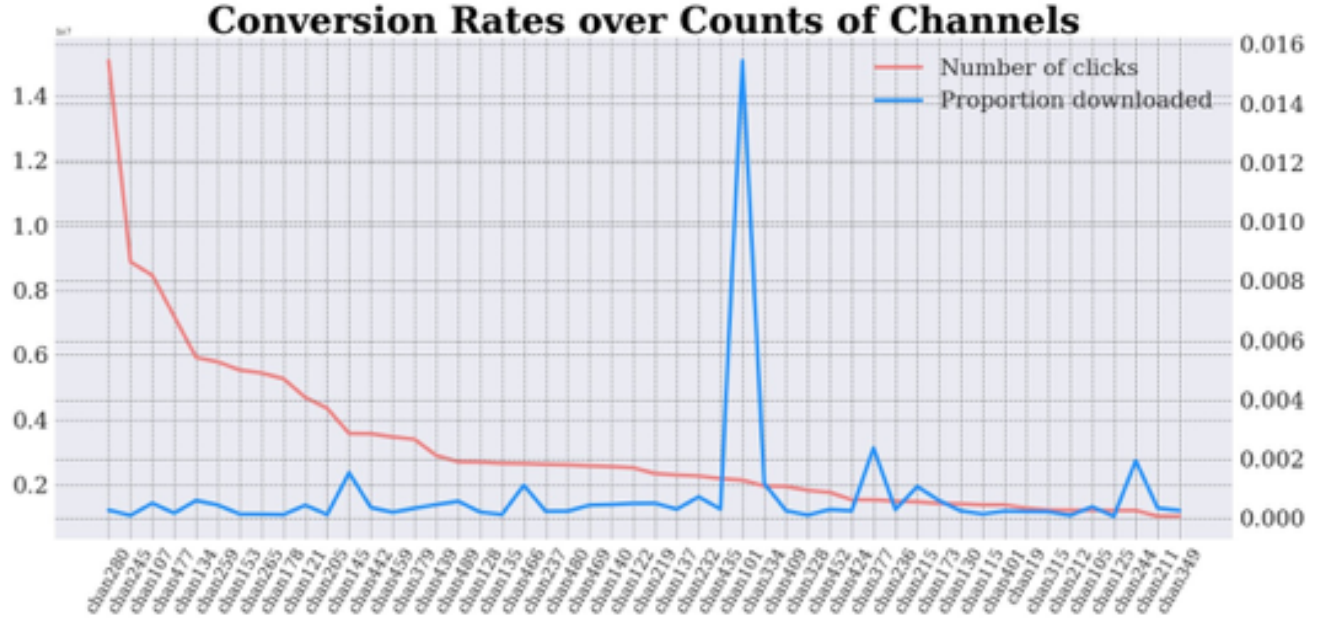

Figure 9. Conversion rates over counts of `channel`.

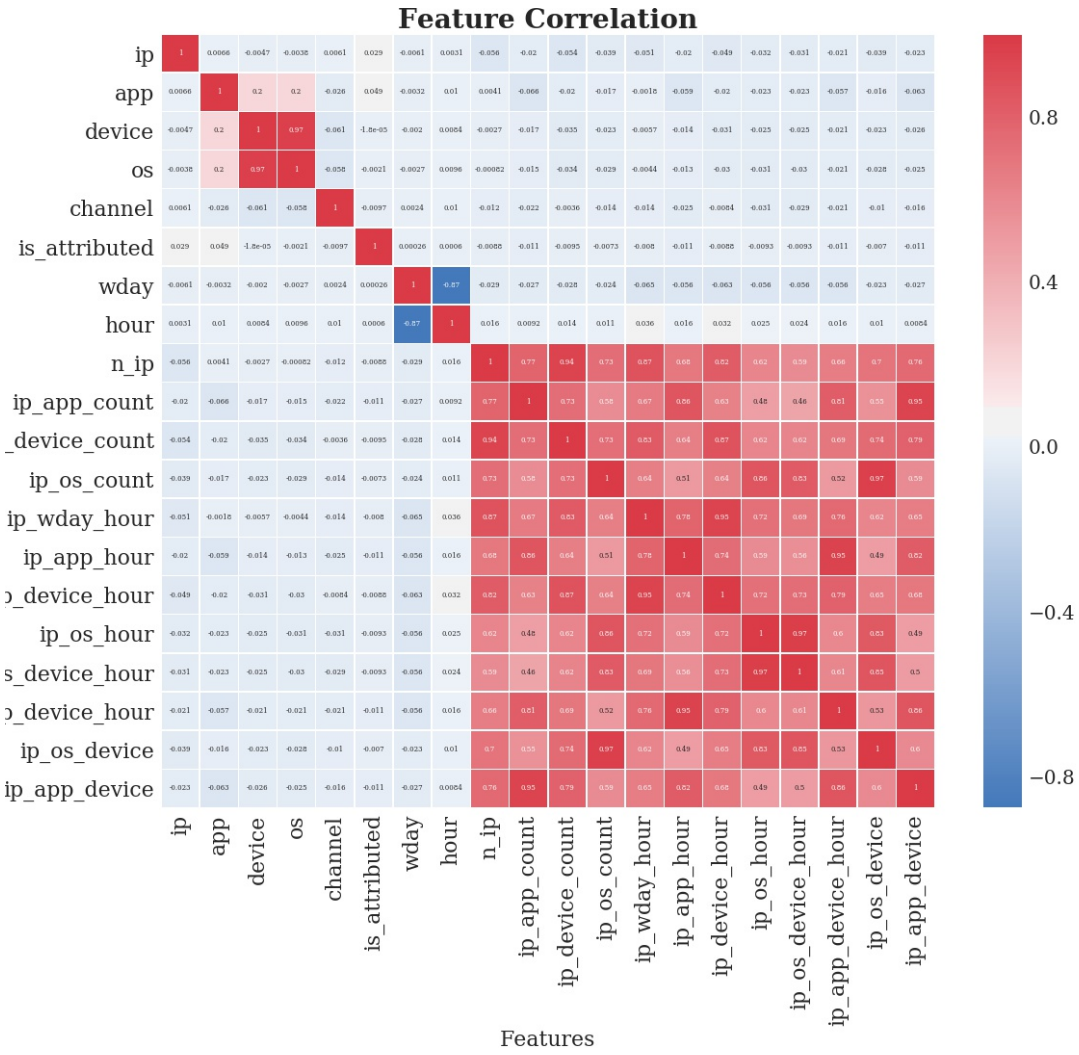

Figure 10. Correlation matrix of features used in the final predictive model.