Mobile App Ads Click Fraud Detection

Zili Chen, Chen Liang, Jinhao Jiang, Bowen Yang, and Jingyang Sui

{zchen688, cliang73, jjiang323, yangbowen, jysui}@gatech.edu

1 Introduction

1.1 Motivation

Nowadays, companies' online advertisement fee is allocated based on users' click data. However, more and more companies are suffering from fraudulent clicks. Our project focuses on detect fraud clicks on mobile app advertisements which induces clicks but do not end up installing apps.

1.2 Problem Definition

With data analysis methods and interactive visualization, we will predict whether a user will download an app after clicking a mobile app advertisement and visualize our result to help companies reduce charges.

2 Survey

Ad-click fraud is now considered one of the biggest scam in the cyberworld with an estimated fraud click rate of 10% - 20%. The fact that site owners benefit a lot from website ad-clicks leads to the emergence of fraudsters [1]. An examination on activities of ZeroAccess showed that it generates on the order of one million fraudulent clicks per day, with the overall ecosystem revenue diverted by the botnet's activity on the order of $100,000, all of which demonstrate the necessity of fraud detection techniques [2].

2.1 Historical Algorithms

Click fraud detection involves detecting a network-orientated service based on the number of accessible users [3]. One earlier algorithm is Bluff Ads [4], which increased the effort level for click fraudsters and tested the legitimacy of individual clicking behavior to identify fraudulent clicks. However, Bluff Ads had scalability limitations.

In recent years, learning-based prediction models became popular. One work proposed an online advertising algorithms [5] that could detect duplicate clicks over decaying window models (e.g. sliding windows and jumping windows) based on timing bloom filters. Another paper introduced an ensemble learning approach [6], which proposed novel feature selection and embedding techniques. It identified common patterns (e.g. clicking time intervals, IP addresses) by analyzing related features. However, it failed to examine whether the features crafted by merging multiple attributes are sufficiently representative. Google AI explored the real-world challenges of fraud detection and proposed several tricks (e.g. memory saving, model selection and assessment). It carried out case studies of Follow the Regularized Leader (FTRL) algorithm [7]. A work by FAIR introduced an approach to merge gradient boosting decision tree (GBDT) and logistic regression [8]. Experiments showed that identifying correct feature was the key to success. However, it also showed that GBDT was less suitable for online learning setting. Some other works proposed open-source frameworks for GBDT training [9] [10] that our project can take advantage of. Their scalability allowed for efficient training on sparse data while sacrificing the accuracy.

2.2 Network Systems and Mobile Applications

A paper in 2010 called for combining academic algorithms with the practical situations and real datasets [11]. Since then, many works integrated the algorithms into pipelined network systems for industry usage. A real-time system was proposed to block click fraud based on the collaboration between server side and client side [12]. Another work created a system for handling click fraud in mobile environments [13]. This system could determine the validity of the user requests and flag those invalid ones as potential fraud. Another real-time ad-click fraud detection system analyzed the click record data and assigned a score for each record based on the quality of the click [14]. However, it required a large dataset for evaluation.

Some other works developed novel usages of existing algorithms on click fraud detection applications. An application in 2014 allowed users to detect click fraud activities issued by clickbots on advertiser side [15]. It consisted of proactive functionality testing and passive browsing-behavior examination. The first part asked the client to prove his/her authenticity by pushing a JavaScript script; the second part monitored the users' behaviors.

3 Proposed Method and Intuition

3.1 Data Preprocessing

The raw dataset is from the data platform of TalkingData, China's largest independent big data service. The dataset provided on the website consists of both training and test sets. The raw training set consists of 1000,000 click instances and raw test set consists over 1000 click instances. Since the size of raw training set is sufficiently large and the raw test set includes no ground truths, we decide to split the raw training set for both training and evaluation purpose in our project.

The dataset is a .csv file, each row of which is a click instance associated with 7 attributes. The attributes are: user IP that induces the click instance ("IP"), the mobile app("app"), mobile device("device") and OS of the device("OS") that the click is occurring on, channel number("channel"), the time-stamp the click is induced("click_time") and the time-stamp the user download the app if there is one ("attributed_time"). The binary label we try to predict is "is_attributed", which is either 1 or 0 indicating whether this user download the app or not after clicking the ads.

We first remove all extreme abnormal values which maybe introduced due to incorrect origin data input. Then we convert the "click_time" data-type from string to date-time, and further convert it to numerical values by calculating the difference between current "click_time" with minimal "click_time" in unit of seconds. Since most values in "attribute_time" column are null, we perform feature selection by removing "attribute_time" column. In contrast, the other columns not only have meaningful rows, but also show significantly distinguished effects on final classification result (See section 4.Q1 and section 4.Q2). Hence, we keep the other 6 feature attributes. In addition, we unify data format for our dataset. We first obtain the standard name from each company or app's official website, then we use regex library to help find and replace different expression with the standard ones.

3.2 Model Training

Negative Down Sampling
Negative down sampling is a strategy to solve

the training set imbalance problem when the number of negative samples is much larger than the number of positive training samples. The goal is to retain or amplify the positive samples, while reduce the number of negative samples in the training set, such that the final amount of positive and negative samples are comparable.

There are some intuitions why we design and adopt the negative down sampling method for model training. During data analysis, we notice the ground truth labels of above 95% click instances are 0, which indicates the majority of ad-clicks does not contribute to app downloading. If trained with this original dataset, the model would learn from unbalanced positive and negative information. Intuitively, the model trained on unbalanced data would have a strong bias to the prior distribution of labels, and therefore will perform badly on the prediction of minority samples, which in our case, the positive samples. By using negative down sampling, we reveal more positive samples information to the model given a fixed memory limit.

Gradient Boosting Decision Trees (GBDT)
GBDT produces a prediction model as an ensemble of weak learners (i.e. decision trees) stage-wisely. GBDT is more generalizable than other boosting methods since it can optimize over any differentiable objectives. GBDT evaluates the information gain for each feature it splits on and marginalizes the feature with low information gain (i.e. features with insignificant contribution to the classification).

There are several reasons we choose GBDT as one of our classifier candidates for our supervised binary classification task. The first reason is: GBDT makes splits up to the max depth by algorithm, and then starts pruning the tree backwards and remove splits beyond which there is no positive gain. Compared with other gradient boosting machines, which stop splitting a node when it encounters a negative loss in the split, GBDT can usually explore deeper in space. Considering we have multiple attributes and each of them has high variance, the model may obtain more information by deepening the tree. Another reason is: in GDBT, the leaf weight values of the trees can be regularized using popular regularization functions (i.e. L2 and L1), which effectively reduces overfitting induced by exploring too deep. The third reason is: most of the feature attributes assume discrete values. GBDT can handle discrete values effectively by using CART.

Multilayer Perceptron (MLP)
MLP is a three-layer neural network consists of an input layer, a hidden layer and an output layer. Each layer is fully-connected with the next layer. MLP can model non-linearity by using activation functions between each layer, which allows the model to learn more complicated relationships between input and output.

We choose MLP as one of our classifier candidates because it is a powerful tool in dealing with non-linearity in data. Consider the size of training data is not huge enough for deep neural networks, we choose to use the shallow model. We use Relu for non-linear activation, cross-entropy as loss function.

Other Learning Algorithms
We also explore the performance of Supported Vector Machine (SVM) and Random Forest(See section 4.Q6).

3.3 User Interface

In this project, the client will submit a batch or click data into our model and the model will give the fraud rate for each click in the batch. After getting the prediction result, we will merge the result together with original dataset to form our new dataset. Then we will build a restful api using web framework like flask to help front end fetch data from our backend new dataset. Since the total fraud rate is very high, to visualize the input batch data, we plan to list the first 20 non-fraud clicks and some of their important attributes like click time, ip address and os. Clients can also choose to visualize the batch data in other ways. For instance, show the fraud rate group by ip address, os, click time period of a day, etc. These requests can be also sent by our restful api to backend and the group data can be handled and processed by the backend and passed back in response body to frontend. And we plan to utilize D3 as the tool of plotting.

As mentioned in the proposal method, we plan to visualize the batch query result by D3. Once the back-end pass the fraud rates, we will plot the statistics of the relation between fraud rates and each attribute. Client will get an overview of each figure and can get the detailed plots with interaction.

To be noted, the click time here represents the relative time regarding the earliest click time in training set which is 2017 11. 7 00:00:00, because the model is trained on entire batch instead of certain instance,ãĂĆTherefore, when using our fraud click detector, user to need self-calculated the relative time value (number only) and input it into corresponding field.

4 Experiment & Evaluation

Question 1: Is there any relationships between conversion rate and click features?

Observations: We looked into the conversion rate over counts of different IPs and found that conversion rate is not connected to the IP frequencies. However, we found 22 IP address trigger clicks more than 100 times, identified as suspect fraud clicks. We then explored the distribution of their features like App, device, OS and channel and found fraud clicks mainly happen in certain small range of App, device, OS and channel. We display the cumulative distribution function (CDF) of OS as an example:

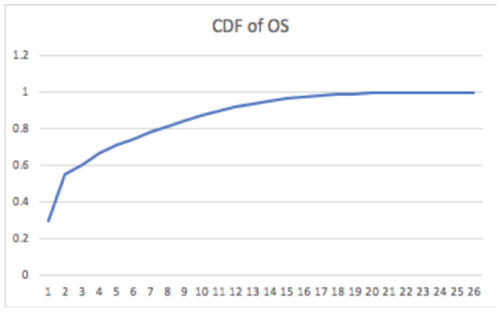


Figure 1

From the figure above, we can also tell that top 10 operating system make up about 68.8% of clicks produced by those top 22 IP addresses, this also implies that majority of suspect fraud clicks happens on few certain operating systems.

Similarly, we found that top 10 apps make up about 81.6% of clicks produced by those 22 IP addresses, this implies that most suspect fraud clicks are concentrate on certain small range of apps.

We also found that top 2 device makes up about 97.68% of total suspect fraud clicks, especially the type 1 device, this implies the device type might be an important factor for determining whether fraud click will be generated.

But we found the distribution of suspect fraud click among channels are more evenly compare with the other three attributes, this suggests that channels may not be a significant distinguisher for determine fraud clicks.

Question 2: Is there any time pattern of these clicks?

Observations: We plotted the hourly click and conversion frequency to gain a brief idea of time pattern. Although there is no clear pattern in the hourly conversion ratio, there is a cycle pattern of the hourly number of clicks. The click frequency is roughly the same every day and always decreases to the lowest point at the end of day. The figures are shown below.

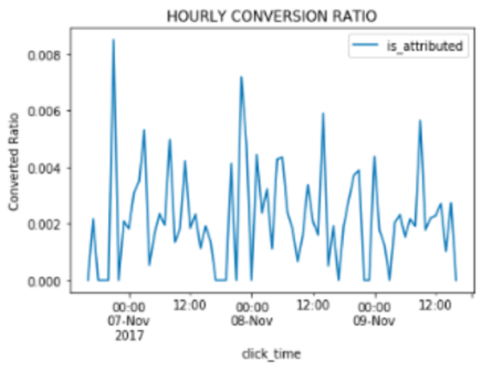


Figure 2

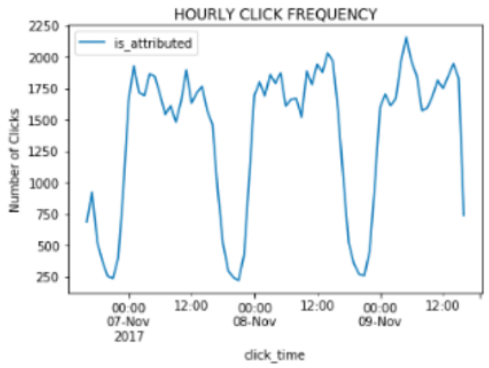


Figure 3

Question 3: How to generate a balanced dataset using negative down sampling?

Table 1 shows the statistics for our generated training and testing sets.

We construct two training sets, one is generated with negative down sampling (*Set A*), and another is generated without negative down sampling (*Set B*). In order to make sure the two training sets are of the same size, we follow the steps below. To obtain *Set A*, we perform negative down sampling on all available training data. We kept all positive samples, and randomly select without replacement negative samples from the rest until the number of positive samples and selected negative samples are identical. Then we randomly shuffle the negative and positive samples in *Set A*. To obtain *Set B*, we randomly select samples with equal size of *Set A* without replacement on all available training data. Since the total amount of all available training data are huge, we perform the above steps batch by batch sequentially. Finally, we end up with two training sets with equal size of around 40000 as shown in Table 1.

We construct five test sets of different sizes with train-test split of roughly 4:1. Note that the original dataset contains the ground truth labels for these instances for evaluation purpose, but we choose not to do negative sampling on test sets. The reason is that the test samples the users provide us in practice would follow the original data distribution, which is in nature an unbalanced distribution.

		Size
Training	Train/Val Sets (10-fold cross validation)	38452
Testing	Test Set 1	9372
	Test Set 2	11149
	Test Set 3	12782
	Test Set 4	14012
	Test Set 5	14306

Table 1: Dataset Information

Question 4: What types of metrics we should choose to evaluate model performance on highly unbalanced dataset?

We choose Area Under Receiver Operating Characteristic curve (AUC) as our evaluation metrics. AUC score measures the two-dimensional area underneath Receiver Operating Characteristic curve (ROC). ROC is a graph showing True Positive Rate (TPR) v.s False Negative Rate (FNR) of a classification model at different classification thresholds. An excellent model has AUC score near 1, which means it has good measurement of separability of positive and negative classes. TPR and FNR are

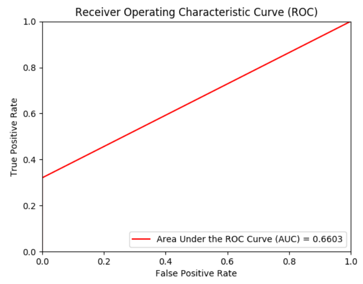


Figure 4: GDBT training ROC using unbalanced dataset

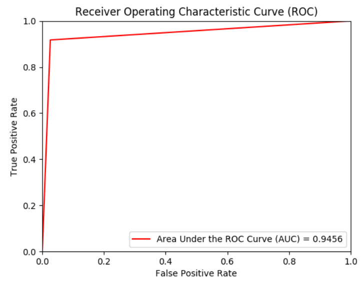


Figure 5: GDBT training ROC using balanced dataset

defined as the following:

$$\text{True Positive Rate(TPR)} = \frac{\text{True Positive(TP)}}{\text{True Positive(TP)} + \text{False Negative(FN)}}$$

$$\text{False Negative Rate(FNR)} = \frac{\text{False Positive(FP)}}{\text{False Positive(FP)} + \text{True Negative(TN)}}$$

Note that we do not evaluate our model under classification accuracy metrics. For highly unbalanced dataset as in our case, high classification accuracy under a fixed threshold does not necessarily ensure sufficient class separation ability of a classifier. The classifier can easily get above 95% classification accuracy if it predicts all test samples as the dominant class labels, and give wrong prediction for all minor test samples. Hence, we do not use classification accuracy under our problem setting.

We also show the precision and recall scores in the experiments below.

Question 5: Is negative sampling effective?

Table 2 compares the training and testing performance between the model trained on training set generated using negative down sampling (*Model A*) and the model trained on training set generated without using negative down sampling (*Model B*).

From Table 2, we can observe that *Model A* clearly out-performs *Model B* under a fixed type of classifier. All the classifiers using *Model B* obtained a training and testing score of around $0.50 \sim 0.60$, which indicates *Model B* barely learn useful information but random guess.

Figure 4 and Figure 5 show 2 visualization examples of training ROC generated by GBDT model on balanced and unbalanced datasets.

Table 3 shows examples of precision-recall statistics of testing performance on test set 1 and 3. We find that the recall for the positive class is in the order of 1×10^{-2} when testing with *Model B*, which indicates that, given unbalanced data, *Model B* tends to assign dominant class labels for all test samples, which is in agree with our earlier assumption that the class conditional distribution prior is too strong. In comparison, *Model A* gives around 0.9 recall for positive class, which proves negative down sampling approach is effective.

Question 6: Which type of classifier should we choose?

Table 2 shows a comparison of training and testing performance among 4 different classifiers. We fine-tuned the parameters of GBDT, Random Forest and SVM using 10-fold cross validation. We normalized the training and testing data for SVM and MLP for faster training speed. For MLP, we use constant learning rate of 0.01, Adam optimizer with momentum, L2 regularization and max iteration of 200. GBDT parameters are shown in Table 5.

Method	GBDT		Random Forest		SVM		MLP	
Trained Model	*Model A*	*Model B*	*Model A*	*Model B*	*Model A*	*Model B*	*Model A*	*Model B*
Training AUC	0.9456	0.6603	0.9882	0.8962	0.7577	0.5000	0.8630	0.5000
Testing AUC (Averaged)	0.9012	0.5542	0.8869	0.5484	0.6573	0.5000	0.6975	0.5000

Table 2: Training performance is obtained by inferencing the fine-tuned model (10-fold cross validation) on the corresponding training set. Testing performance is the average score obtained from all 5 test sets.

From Table 2, we note that GBDT and random forest achieve above 15% higher training and testing performance than other classifiers. Besides, we can see that GBDT performs better than random forest with fine-tuned parameters. The possible reason can be: in terms of training objective, GBDT tries to add new trees that compliments the already built ones. This normally gives better accuracy even with less trees. Furthermore, we can observe that the MLP classifier achieves a training score higher than the testing score for around 20%, which indicates that the MLP classifier is overfitting on the training set even after regularization. Simultaneously, the training score of the MLP classifier is less than 0.90, which means it underrepresents the training data. It is possibly because the amount of training data is inadequate for such complex model. For SVM, both the training and testing performance are not satisfying, which indicates that the model is too simple for this task.

Hence, we choose GBDT as our final model. Table 4 shows the detailed performance of the final GDBT model on training and test sets. The training AUC after 10-fold cross validation is 0.9456. The average testing AUC is 0.9012.

Figure 4 shows the training ROC for our final GBDT model. Figure 6 - 10 show the testing ROC on 5 test sets for our final GBDT model.

Table 5 shows the final hyper-parameters we select for GBDT.

Question 7: What can be a good way visualize the clicks?

Result: We observe that the clicks are highly biased; most of the clicks are fraudulent. As a result, if we directly try to visualize the whole dataset, it can be quite confusing. Considering that the user is more interested in some outstanding characteristics of fraud clicks, we decide to only visualize the top occurred instance of each feature, which can help clients

Test Set	Label Type	Model A		Model B		
		Precision	Recall	Precision	Recall	# Instances
1	0	0.9997	0.9371	0.9983	0.9999	9355
	1	0.0233	0.8235	0.5000	0.0588	17
3	0	0.9997	0.9332	0.9966	0.9998	12732
	1	0.0513	0.9200	0.6667	0.1200	50

Table 3: Precision-recall statistics for several testing performance using Model A and Model B

		Precision (WAVG)	Recall (WAVG)	F1-score (WAVG)	Area Under ROC Curve
Training	Train/Val Sets	0.9470	0.9456	0.9456	0.9456
Testing	Test Set 1	0.9979	0.9369	0.9657	0.8803
	Test Set 2	0.9967	0.9291	0.9608	0.8296
	Test Set 3	0.9960	0.9332	0.9619	0.9266
	Test Set 4	0.9962	0.9324	0.9615	0.9572
	Test Set 5	0.9970	0.9274	0.9597	0.9125
	Average	0.9968	0.9318	0.9619	0.9012

Table 4: Training performance is obtained by inferencing the fine-tuned GBDT model (10-fold cross validation) on the training set generated by using negative sampling. Testing performance is obtained by inferencing the model on 5 test sets. *WAVG: Weighted Average

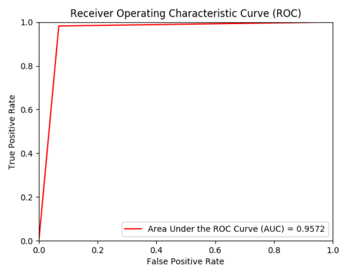


Figure 9: GBDT ROC: test set 4

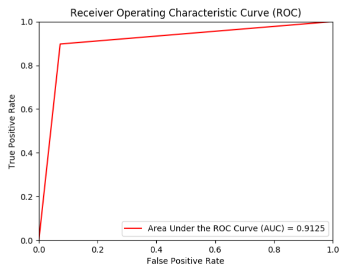


Figure 10: GBDT ROC: test set 5

Objective	Sigmoid
Booster	Gradient Boosting Tree
No. of Estimators	100
Learning Rate	0.1
Min Child Weight	4
Max Depth	20
Regularization Alpha	0.01
Gamma	0.9
Subsample	1
Colsample Bytree	0.9

Table 5: Hyperparameter setting for GBDT

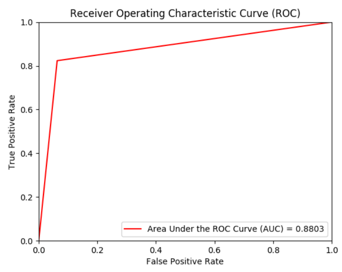


Figure 6: GBDT ROC: test set 1

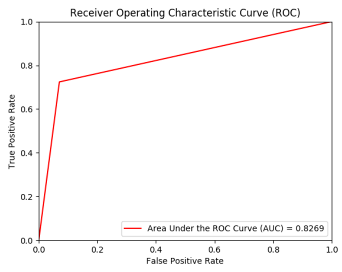


Figure 7: GBDT ROC: test set 2

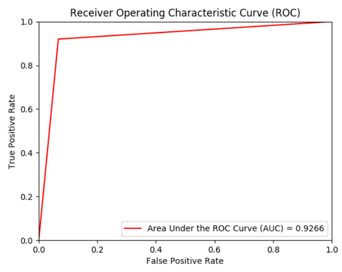


Figure 8: GBDT ROC: test set 3

to quickly get some idea about what kind of clicks is likely fraudulent.

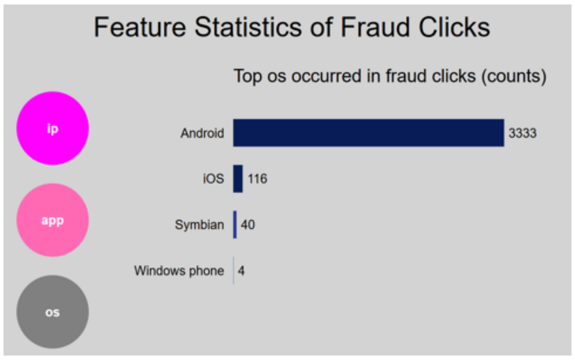
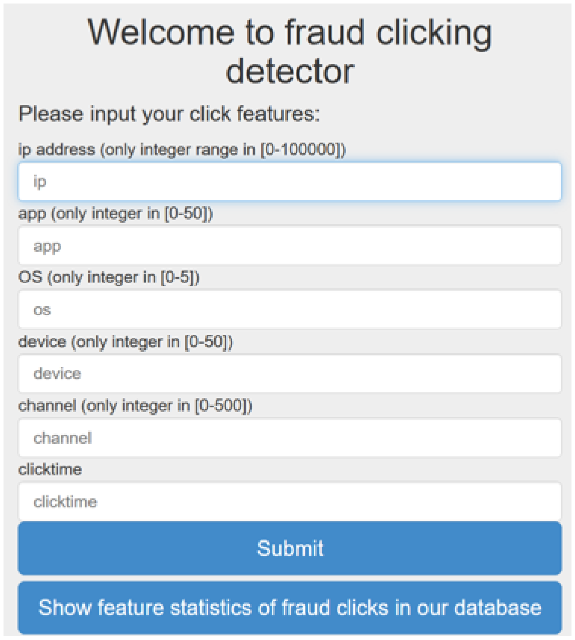


Figure 11: Bar Chart of Statistics


Welcome to fraud clicking detector

Please input your click features:

ip address (only integer range in [0-100000])

ip

app (only integer in [0-50])

app

OS (only integer in [0-5])

os

device (only integer in [0-50])

device

channel (only integer in [0-500])

channel

clicktime

clicktime

Submit

Show feature statistics of fraud clicks in our database

Figure 12: User Interface

Evaluation: In this project, we provide the user an API that asks the user to provide the features needed to predict fraud rate. For front end design, we use D3 to draw all the figures and raw JavaScript to process data. For back end design, we deploy a local service written with python framework flask. In the back end, the service calls our trained model to predict the fraud rate. Client can also access our statistics about the top occurred values of each feature in fraud clicks. A typical workflow is shown below:

(1) User inputs the values of the queried click feature (OS, device, etc.) for a single click instance;
(2) User hits the "submit" button to pass the input values to backend;
(3) Backend receives the feature values of the click instance and call our trained model, which predicts a binary label indicating whether the instance is a fraud click;
(4) Backend sends the result back to front end and updates the frontend UI to show the prediction result to the user.

If user go to the statistics page, an interactive visualization is provided. User can move mouse on the interested feature and a bar chart showing the top occurred values of that feature among fraud clicks appears. Note that the prediction statistics we shown in our project are the prediction result from test set 3. For user who want to submit their own batch, they can wrap their data in a pickle file and submit to the backend server directly.

5 Discussions & Conclusions

From data analysis, we found that App, device, OS, channel these four features has a significant influence on final predict result, and we discovered that there is a cycle pattern of the hourly number of clicks.

We build a robust model based on GBDT that can perform well on highly unbalanced dataset, and achieves an average testing AUC score of over 0.90.

For clearance and emphasis on the critical points that clients interested in, we provide a simple UI that help the client to detect fraudulent clicks and give clients an overview of our statistics that helps clients quickly figure out some of the top fraud-prone characteristics.

6 Work Division

In this project, all team members have contributed similar amount of effort. Yang and

Chen are responsible of data cleaning, data preprocessing and analysis. Liang is in charge of model training and testing. Sui works on visualization and front-end design. Jiang takes control of Progress and front-end design.

References

[1] Nir Kshetri. The economics of click fraud. *IEEE Security & Privacy*, 8(3):45–53, 2010.

[2] Paul Pearce, Vacha Dave, Chris Grier, Kirill Levchenko, Saikat Guha, Damon McCoy, Vern Paxson, Stefan Savage, and Geoffrey M Voelker. Characterizing large-scale click fraud in zeroaccess. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 141–152. ACM, 2014.

[3] Juha Salo and Ahti Muhonen. Method and apparatus for detecting click fraud, August 17 2010. US Patent 7,779,121.

[4] Hamed Haddadi. Fighting online click-fraud using bluff ads. *ACM SIGCOMM Computer Communication Review*, 40(2):21–25, 2010.

[5] Linfeng Zhang and Yong Guan. Detecting click fraud in pay-per-click streams of online advertising networks. In *Distributed Computing Systems, 2008. ICDCS'08. The 28th International Conference on*, pages 77–84. IEEE, 2008.

[6] Kasun S Perera, Bijay Neupane, Mustafa Amir Faisal, Zeyar Aung, and Wei Lee Woon. A novel ensemble learning-based approach for click fraud detection in mobile advertising. In *Mining Intelligence and Knowledge Exploration*, pages 370–382. Springer, 2013.

[7] H Brendan McMahan, Gary Holt, David Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, et al. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1222–1230. ACM, 2013.

[8] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9. ACM, 2014.

[9] Richard Oentaryo, Ee-Peng Lim, Michael Finegold, David Lo, Feida Zhu, Clifton Phua, Eng-Yeow Cheu, Ghim-Eng Yap, Kelvin Sim, Minh Nhut Nguyen, et al. Detecting click fraud in online advertising: a data mining approach. *The Journal of Machine Learning Research*, 15(1):99–140, 2014.

[10] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[11] Kenneth C Wilbur and Yi Zhu. Click fraud. *Marketing Science*, 28(2):293–308, 2009.

[12] Li Ge and Mehmed Kantardzic. Real-time click fraud detecting and blocking system, November 1 2007. US Patent App. 11/413,983.

[13] Arvind Gupta, Ashutosh Tiwari, Gopalakrishnan Venkatraman, Dominic Cheung, Stacy R Bennett, and Douglas B Koen. Mobile click fraud prevention, August 5 2014. US Patent 8,799,069.

[14] Mehmed Kantardzic, Chamila Walgampaya, Brent Wenerstrom, Oleksandr Lozitskiy, Sean Higgins, and Darren King. Improving click fraud detection by real time data fusion. In *Signal Processing and Information Technology, 2008. ISSPIT 2008. IEEE International Symposium on*, pages 69–74. IEEE, 2008.

[15] Haitao Xu, Daiping Liu, Aaron Koehl, Haining Wang, and Angelos Stavrou. Click fraud detection on the advertiser side. In *European Symposium on Research in Computer Security*, pages 419–438. Springer, 2014.