

CS6350 Final Report: TalkingData AdTracking Fraud Detection

Hantao Zhang, Tianqi Zhu, Yibo Hu, ZeqingLi

April 27 2018

1. Introduction

Fraud risk is everywhere and it happens at an overwhelming volume on mobile since smart mobile devices widely used. Because of companies that advertise online, user may click fraud during using an App, which resulting in misleading click data and wasted money. By simply clicking on the ad, Ad channels can drive up costs at a large scale. China is the largest mobile market in the world with over 1 billion smart mobile devices in active use every month, therefore suffers from huge volumes of fraudulent traffic.

Losses from fraud greatly threatening your business's existence are relatively rare, however, they do happen. Taking Target for instance, near catastrophic failures can happen overnight in the networked digital world especially. The consequences of large failures causing by Fraud risk can impose financial, reputational, loyalty, and other brand-related costs, which will persist for a long time [1].

A short discussion of how it fits into related work in the area is also desirable. Summarize the basic results and conclusions that you will present [2].

The main approach of the analysis and detection of this problem are listed below.

1. Build pipelines that contain data pre-processing and building machine learning models.
2. Use cross-validation to select best models which will be evaluated on area under the. ROC curve between the predicted probability and the observed target.
3. Cover most classification methods from Spark MLlib library: Decision trees, Random Forest, GBT, and Naïve Bayes which performed an exhaustive study of which classification algorithms achieved the greatest accuracy when used as the base-level classifiers for the model [4].
4. Apply XGBoost with Spark to achieve a better ranking for the Kaggle competition.
5. The area under the ROC curve (AUC) value applied with Naïve Bayes and XGBoost (training dataset and 1 million clicks and 10 million clicks) are presented.

2. Related work

The public kernel for the AdTracking Fraud Detection Challenge shows most solution are written in Python and R with a blend of advanced machine learning models. Out of the highest leaderboard scores models, LightGBM and XGBoost are two techniques achieving the best [7].

Gradient boosting tree is one technique that performs very well in many machine learning applications in practice. One reason we adopt gradient boosting tree for our ad click fraud detection challenge is that it has been incorporated into real-world production pipelines for ad click through rate prediction [5]. On top of the gradient boosting tree technique, Chen and Guestrin built a scalable tree boosting system called XGBoost [6]. XGBoost has been proved to be success in many machine learning scenarios. For example, it has been widely recognized in many data mining and machine learning challenges such as Kaggle and KDD, and it has been the best solution in a number of challenge tasks: store sales prediction; high energy physics event classification; web text classification; customer behavior prediction; motion detection; ad click through rate prediction; malware classification; product categorization; hazard risk prediction; massive online course dropout rate prediction [6]. XGBoost wins its popularity by virtue of outstanding performance, the authors claim that it runs “more than ten times faster than existing popular solution on a single machine and scales to billions of examples in distributed or memory-limited settings”.

3. Dataset description

Kaggle provided a dataset more than 7G which covered approximately 200 million clicks over 4 days [3]. The training data fields are at below:

1. Each row of the training data contains a click record, with the following features.
2. ip: ip address of click.
3. app: app id for marketing.
4. device: device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, etc.)
5. os: os version id of user mobile phone
6. channel: channel id of mobile ad publisher
7. click_time: timestamp of click (UTC)
8. attributed_time: if user download the app for after clicking an ad, this is the time of the app download
9. is_attributed: the target that is to be predicted, indicating the app was downloaded.

The test data is similar, with the following differences:

10. click_id: reference for making predictions

11. is_attributed: not included

4. Pre-processing techniques

We first used tools like databricks and jupyter notebook to better understand the data. The schema of the data was shown as below:

```
-- ip: integer (nullable = true)
-- app: integer (nullable = true)
-- device: integer (nullable = true)
-- os: integer (nullable = true)
-- channel: integer (nullable = true)
-- click_time: timestamp (nullable = true)
-- attributed_time: timestamp (nullable = true)
-- is_attributed: integer (nullable = true)
```

Step 0. Check the data

“is_attributed” was the label of our model and we renamed it for convenience. We checked the data was clean and we extracted time informations like year, month, day and hour from the timestamp column “click_time”. It was obvious that these time features affect the total number of clicks and the frequency of clicking ads. Some analysis pictures of the training sample are shown:

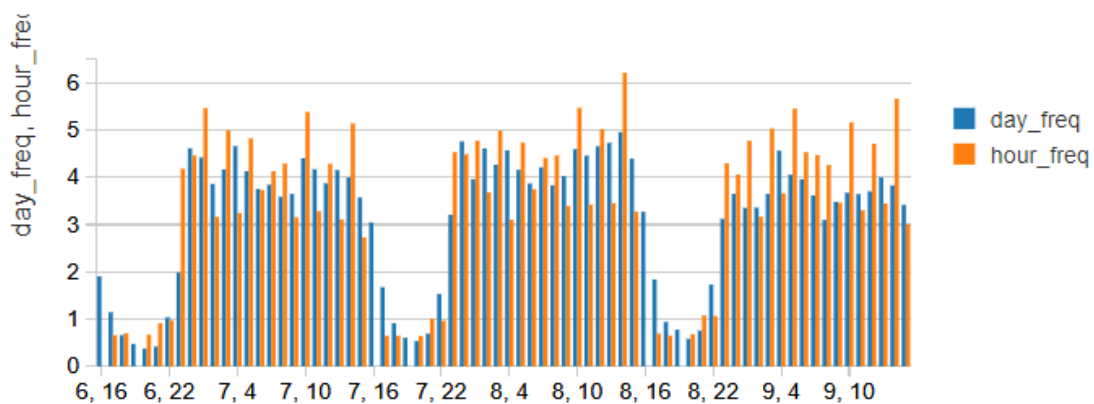


Figure 1 The frequency of clicking ads per day and hour

We ignored year and month because all the clicks happened in November 2017. We also deleted “attributed_time” because it can’t help us predict the label.

The distribution of the data was biased on other features such as “app” and “channel”, as shown below.

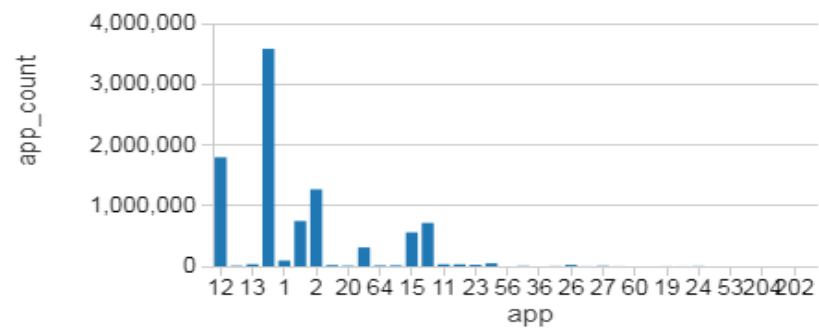


Figure 2 The total counts of clicks per app

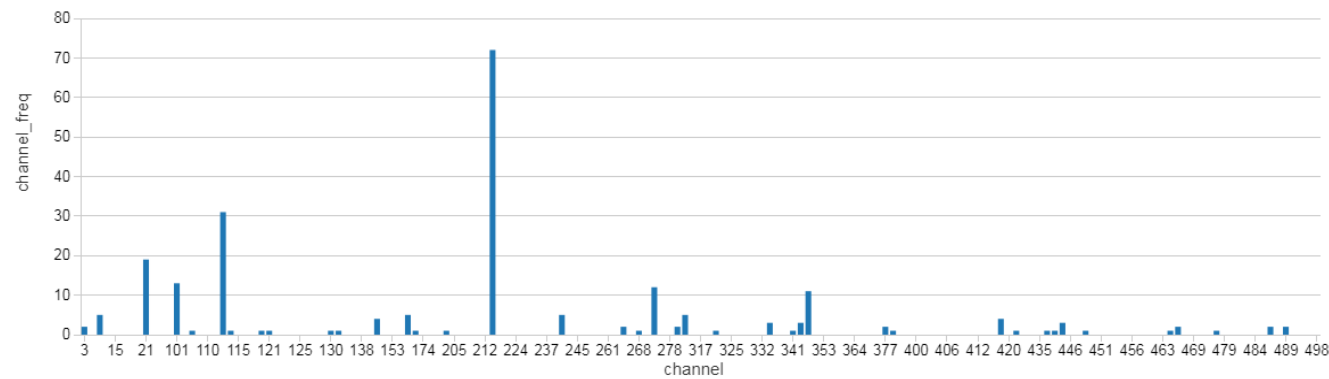


Figure 3 The frequency of clicking ads per channel

Step 1 Feature engineering

We added more features based on group-by aggregation. These features included total clicks for ip-day-hour, total clicks for ip - hour - app etc.

app	day	hour	device	ip	os	channel	label	ip_h_dev	ip_h_app	ip_h_os	ip_day_h	ip_os_dev	nipApp	app_day_h_dev
9	7	16	1	9	13	244	0	1	1	1	1	1	1	119
11	7	1	1	10	22	319	0	1	1	1	1	1	1	22
12	7	7	1	10	19	140	0	1	1	1	1	1	1	198
18	8	11	1	10	13	107	0	1	1	1	1	1	1	153
14	8	9	1	19	16	379	0	1	1	1	1	1	1	88
2	9	3	1	20	16	469	0	1	1	1	1	1	1	200
12	8	0	1	20	13	277	0	1	1	1	1	1	1	218
18	8	8	1	20	19	107	0	1	1	1	1	1	1	85
2	7	11	1	25	22	117	0	1	1	1	1	1	1	226

Figure 4. final features after

Step 2 Implemented a pipeline for pre-process data

First we noticed that "ip", "app" etc to be categorical features. These categorical features were converted into StringIndexer and finally to One hot vectors

<	ip_Vec	app_Index	app_Vec	device_Index	device_Vec	os_Index	os_Vec	channel_Index	channel_Vec
3 (33060,[19062],[1...]	3.0 (159,[3],[1.0])	0.0 (96,[0],[1.0])	1.0 (130,[1],[1.0])	43.0 (160,[43],[1.0])	63.0 (160,[63],[1.0])	24.0 (160,[24],[1.0])	2.0 (160,[2],[1.0])		
3 (33060,[7147],[1.0])	11.0 (159,[11],[1.0])	0.0 (96,[0],[1.0])	4.0 (130,[4],[1.0])	63.0 (160,[63],[1.0])	24.0 (160,[24],[1.0])	2.0 (160,[2],[1.0])			
3 (33060,[7147],[1.0])	1.0 (159,[1],[1.0])	0.0 (96,[0],[1.0])	0.0 (130,[0],[1.0])	24.0 (160,[24],[1.0])	2.0 (160,[2],[1.0])				
3 (33060,[7147],[1.0])	5.0 (159,[5],[1.0])	0.0 (96,[0],[1.0])	1.0 (130,[1],[1.0])	2.0 (160,[2],[1.0])					

Figure 5. One hot vectors (partial) exacted from categorical features

By concatenating these one-hot vectors and the other numeric features into an assembler, we got the final label-features.

label	features
0	(33640,[19062,330...])
0	(33640,[7147,3307...])
0	(33640,[7147,3306...])
0	(33640,[7147,3306...])
0	(33640,[17756,330...])
0	(33640,[8189,3306...])
0	(33640,[8189,3306...])
0	(33640,[8189,3306...])
0	(33640,[25875,330...])
0	(33640,[3827,3306...])

Figure 6. The final label-features

Step 3. Applied the pre-process pipeline model to the data

We used this pre-process pipeline to fit our our training data and got the pipeline model.

This pipeline model was saved and then was used to transform on our training data and test data. Finally we got training and testing label-feature vectors.

5. Machine learning models

The AdTracking Fraud Detection Challenge's goal to predict weather a user will download an app after clicking a mobile app advertisement. Majority of the data fields are categorical hence we give Logistic Regression, Decision Tree, and Gradient Boosted Tree, Naive Bayes a try. Besides the model provided by Spark MLlib, we also apply a third-party library XGBoost, which is an enhanced Gradient Boosted Tree technique, to the task.

Logistic Regression. Logistic regression is a popular method to predict a categorical response. It is a special case of Generalized Linear models that predicts the probability of the outcomes. In spark.ml logistic regression can be used to predict a binary outcome by using binomial logistic regression [8].

Decision Tree, Gradient Boosting Tree, and XGBoost. The family of decision tree models are popular method for machine learning tasks of classification and regression. Decision trees are widely used since they are easy to interpret, handle categorical features, extend to the multiclass classification setting, do not require feature scaling, and are able to capture non-linearities and feature interactions [8]. Gradient-boosted tree is a popular classification and regression method using ensembles of decision trees [8]. XGBoost is a more scalable gradient boosted tree and it trains faster than gradient boosted tree in a magnitude of ten [6].

Naive Bayes. Naive Bayes classifiers are a family of simple probabilistic classifier based on applying Bayes's theorem with strong (naive) independence assumptions between the features. In spark.mllib NaiveBayes implements multinomial naive Bayes.

6. Experiments

We implemented a python program based on Boto3 library to upload the huge training set into our Amazon S3 bucket.

Amazon S3 > snowood1 / project

Overview

🔍 Type a prefix and press Enter to search. Press ESC to clear.

[Upload](#)
[+ Create folder](#)
[More ▾](#)
[Versions](#)
[Hide](#)
[Show](#)
[US East \(Ohio\)](#)
[↻](#)

Viewing 1 to 6

<input type="checkbox"/>	Name ↑ ▾	Last modified ↑ ▾	Size ↑ ▾	Storage class ↑ ▾
<input type="checkbox"/>	📁 /	Apr 21, 2018 9:45:00 PM GMT-0500	0 B	Standard
<input type="checkbox"/>	📄 test.csv	Apr 21, 2018 9:52:37 PM GMT-0500	823.3 MB	Standard
<input type="checkbox"/>	📄 train-10M.csv	Apr 25, 2018 1:40:13 PM GMT-0500	408.6 MB	Standard
<input type="checkbox"/>	📄 train-1M.csv	Apr 24, 2018 6:51:03 PM GMT-0500	52.8 MB	Standard
<input type="checkbox"/>	📄 train.csv	Apr 22, 2018 12:57:58 AM GMT-0500	7.0 GB	Standard
<input type="checkbox"/>	📄 train_sample.csv	Apr 22, 2018 3:19:14 PM GMT-0500	3.9 MB	Standard

Figure 7. Our data in the AWS S3 bucket

Trained and tested our data on AWS EMR

Step 1. Trained the pipeline and save

Step 2. Make predictions with a Naive Bayes model

Step 3. Cross validation

It took too much time to train and tuned the whole dataset(7GB) so we selected 9 million clicks training dataset and 1 million clicks to be testing dataset in this project.

Cluster: My cluster 16 sly Terminated Terminated by user request

Summary Application history Monitoring Hardware Events Steps Configurations Bootstrap actions

Add step Clone step Cancel step

Steps

[View all interactive jobs](#) | [View all jobs](#)

Filter: <div>All steps</div> <div>Filter steps ...</div>		4 steps (all loaded) <div></div>					
	ID	Name	Status	Start time (UTC-5) <div></div>	Elapsed time	Log files	Actions
<div><div></div><div></div></div>	s-2YIMJJIZDSMYR	Spark application	Completed	2018-04-26 01:30 (UTC-5)	1 minute	View logs	View jobs
<div>JAR location : command-runner.jar</div> <div>Main class : None</div> <div>spark-submit --deploy-mode cluster --class Evaluation s3://utdallas6350hantao/bigdatafinal/ML-assembly-0.1.jar s3://utdallas6350hantao/NB409out_predicts.parquet/</div> <div>Arguments : s3://utdallas6350hantao/eval_res_NB409</div> <div>Action on failure: Continue</div>							
<div><div></div><div></div></div>	s-2BQSYR78TRM40	Spark application	Failed	2018-04-26 01:24 (UTC-5)	44 seconds	controller syslog* stderr stdout* <div></div>	View jobs
<div><div></div><div></div></div>	s-1LK60CP9KP1ZO	Spark application	Completed	2018-04-26 00:56 (UTC-5)	19 minutes	View logs	View jobs
<div>JAR location : command-runner.jar</div> <div>Main class : None</div> <div>spark-submit --deploy-mode cluster --class NB s3://utdallas6350hantao/bigdatafinal/ML-assembly-0.1.jar s3://utdallas6350hantao/outputPre409M_labelfeatures.parquet/</div> <div>Arguments : s3://utdallas6350hantao/outputPre409M_pipeline_model s3://utdallas6350hantao/outputPre409M_test/ s3://utdallas6350hantao/NB409out</div> <div>Action on failure: Continue</div>							
<div><div></div><div></div></div>	s-2I4GBPOY43QLD	Setup hadoop debugging	Completed	2018-04-26 00:52 (UTC-5)	2 seconds	View logs	View jobs

Solutions of the issues during our experiments

We faced “java.lang.IllegalArgumentException: Size exceeds Integer.MAX_VALUE” issues. This was due to that no Spark shuffle block can be larger than 2GB (Integer.MAX_VALUE bytes) . We added all the dataframes in our programs up to 500 partitions and successfully solved this problem.
























<input type="button" value="Upload"/> <input type="button" value="+ Create folder"/> <input type="button" value="More ▾"/>		US East (N. Virginia) 		
<input type="checkbox"/>	Name 	Last modified 	Size 	Storage class 
<input type="checkbox"/>	 NB409out_predicts.parquet	–	–	–
<input type="checkbox"/>	 NB409out_tests.parquet	–	–	–
<input type="checkbox"/>	 bigdata7G	–	–	–
<input type="checkbox"/>	 bigdatafinal	–	–	–
<input type="checkbox"/>	 eval_res_NB409_auroc	–	–	–
<input type="checkbox"/>	 homeworkresult	–	–	–
<input type="checkbox"/>	 outputPre409M_labelfeatures.parquet	–	–	–
<input type="checkbox"/>	 outputPre409M_pipeline	–	–	–
<input type="checkbox"/>	 outputPre409M_pipeline_model	–	–	–
<input type="checkbox"/>	 outputPre409M_test	–	–	–
<input type="checkbox"/>	 outputsample22_H_labelfeatures.parquet	–	–	–
<input type="checkbox"/>	 outputsample22_H_preprocess_model	–	–	–
<input type="checkbox"/>	 outputsample22_H_preprocess_pipeline	–	–	–
<input type="checkbox"/>	 outputsample22_L_labelfeatures.parquet	–	–	–
<input type="checkbox"/>	 outputsample22_L_preprocess_model	–	–	–
<input type="checkbox"/>	 outputsample22_L_preprocess_model_alpha	–	–	–
<input type="checkbox"/>	 outputsample22_L_preprocess_pipeline	–	–	–
<input type="checkbox"/>	 outputsample53M_labelfeatures.parquet	–	–	–

Figure 8. Output results in the AWS bucket

7. Results and analysis

NB (Naive Bayes): 0.8063

label	features	rawPrediction	probability	prediction
0.0	(43028,[1265,4212...	[-514.93733691830...	[1.0,1.8328889828...	0.0
0.0	(43028,[12213,421...	[-204.02796017630...	[0.99999969519963...	0.0
0.0	(43028,[10008,421...	[-248.28127155615...	[0.96652524930368...	0.0
0.0	(43028,[143,42133...	[-1233.9645309487...	[7.87205902469929...	1.0
0.0	(43028,[8794,4215...	[-162.21839220965...	[0.00268773030482...	1.0
0.0	(43028,[2907,4212...	[-588.75988912998...	[1.0,4.7717320493...	0.0
0.0	(43028,[82,42127,...	[-1258.9924512467...	[0.99999999999999...	0.0
0.0	(43028,[4294,4212...	[-555.70559969935...	[1.0,3.1034534308...	0.0
0.0	(43028,[1956,4212...	[-543.71499976216...	[1.0,5.0744123677...	0.0
0.0	(43028,[432,42146...	[-351.72149008695...	[2.77367425092144...	1.0

only showing top 10 rows

Evaluation: areaUnderROC 0.8062813928960322

Figure 9

LR (Logistic Regression): 0.5504

label	features	rawPrediction	probability	prediction
0.0	(43028,[1,42129,4...	[7.89748222666185...	[0.99962845992857...	0.0
0.0	(43028,[1691,4213...	[7.94913856859494...	[0.99964715845132...	0.0
0.0	(43028,[481,42131...	[7.92692026556066...	[0.99963923403152...	0.0
0.0	(43028,[834,42132...	[7.94249449934966...	[0.99964480717795...	0.0
0.0	(43028,[0,42127,4...	[7.95183906022001...	[0.99964810967670...	0.0
0.0	(43028,[3103,4212...	[7.95496910319177...	[0.99964920900080...	0.0
0.0	(43028,[121,42130...	[7.96954333351252...	[0.99965428267984...	0.0
0.0	(43028,[1889,4213...	[7.94749002955955...	[0.99964657650428...	0.0
0.0	(43028,[634,42134...	[7.93117445270312...	[0.99964076498732...	0.0
0.0	(43028,[19863,421...	[7.93506936329038...	[0.99964216095454...	0.0

only showing top 10 rows

Evaluation: areaUnderROC 0.5503879955290213

Results Summary:

Model	Area Under Curve	
	1 million	10 million
Logistic Regression	0.5504	
Decision Tree	N/A	
GBT	N/A	
Random Forest	N/A	
Naive Bayes	0.8063	0.7853
Xgboost	0.6099	0.7581

Logistic Regression. Obviously this dataset was very biased and not linear separable so logistic regression was totally not fit for this dataset.

Decision Tree, Random Forest, and Naive Bayes. The decision tree model and its relative models, like random forest and Gradient-Boosted Trees took too much time to train --- it was not fit for the huge feature vectors

Naive Bayes: Very fast and achieved a good result without much tuning

XGBoost: Fast and good as well. Potential. More tuning was needed to achieve a better result.

8. Conclusion

Of the algorithms we tested, the Naive Bayes classifier with 30 selected features produced the highest test area under curve of 78.53% for 52 million records. Surprisingly it was better than expected good performer model XGBoost. We think that the high bias of our training data attributed to this result.

We believe that the metric of XGBoost technique can be increased by training on the full data set. We also believe that the metric for both data can be further increased by adopting more features as we only adopt a subset of features due to computation resource limit. The biggest hurdle of accuracy is the lack of real ad click data present in the dataset, further work might be done by testing the model under similar amount of real click and fraud click data.

9. Contribution of team members (sorted by starting time)

Yibo Hu:

Survey and feature engineering

Implemented a preprocess pipeline model, a logistic regression and a naive bayes model

Implemented programs to split and upload big datasets into S3.

Hantao Zhang:

Implemented a decision tree model

Debugged and tested codes on the local machine

Conducted tests of jar files on AWS

Zeqing Li:

Built a github repository for the group's discussions

Implemented final MLlib models such as random forests and GBT

Implemented an Xgboost model

Tianqi Zhu:

Investigated the fitness and limits for each models.

Revised the final report

Reference

- (1) “Why Fraud Prevention Really Matters.” *The Risk Management Blog* / *Lowers & Associates*, 28 Sept. 2017, blog.lowersrisk.com/why-fraud-prevention-matters/.
- (2) *TalkingData AdTracking Fraud Detection Challenge* / *Kaggle*, www.kaggle.com/c/talkingdata-adtracking-fraud-detection.
- (3) *TalkingData AdTracking Fraud Detection Challenge* / *Kaggle*, www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data.
- (4) Geis, Srinivas, Talreja. 2015. CS 229 Final Report: Activity Recognition Using Cell Phones. P. 1-2.
- (5) X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. n. Candela. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, ADKDD’14, 2014.
- (6) T Chen, C Guestrin, Xgboost: A scalable tree boosting system, *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (New York, NY, USA)*, KDD ’16, ACM, 2016, pp. 785–794.
- (7) TalkingData: R LightGBM Tutorial with Model Evaluation, <https://www.kaggle.com/pranav84/talkingdata-eda-to-model-evaluation-lb-0-9683>
- (8) Spark MLlib Classification and Regression. <https://spark.apache.org/docs/latest/ml-classification-regression.html#classification-and-regression>