Hybrid Models for Click Fraud Detection in Mobile Advertising

Chen Wei WEICHEN@COMP.NUS.EDU.SG
School of Computing
National University of Singapore
Computing 1, Computing Drive, Singapore

Dhaval Patel DHAVAL@COMP.NUS.EDU.SG
School of Computing
National University of Singapore
Computing 1, Computing Drive, Singapore

Editor:

Abstract

Advertising plays a vital role in supporting free websites and smart phone apps to target their marketing campaigns to the appropriate customers. Advertisers pay the publisher (typically a website owner) when the ad is clicked (pay per click). However, Malicious publishers generate clicks that do not have genuine interest in the advertisements, which is called "Click Fraud". It results in advertising revenue being misappropriated by clickspammers. It is important to take take active measures to block clickspam today. BuzzCity provides a snapshot of their click and publisher database as dataset for the competition. The goal of this competition is to detect fraudulent publishers. This paper describes the solution of the National University of Singapore team. We exploited a diverse of models (decision tree models, neural network models, models etc). Our final submission is a blend of different learning algorithms. The algorithms are trained consecutively and they are blended together to achieve **62.12%** and **46.15%** in average precision on the validation and testing dataset seperately.

Keywords: click fraud detection, decision tree, neural network, blending, ensemble

1. Introduction

Advertising plays a vital role in supporting free websites and smart phone apps to target their marketing campaigns to the appropriate customers. Advertisers pay the publisher (typically a websites owner) when the ad is clicked (pay per click). However, Malicious publishers generate clicks that do not have genuine interest in the advertisements, which is called "Click Fraud". Click fraud costs online advertisers on the order of hundreds of millions of dollars each year. Incentives for click-spam are linked directly to the flow of money. In online advertising, advertisers pay ad networks for each click on their ad, and ad networks pay publishers (websites or phone apps that show ads) a fraction of the revenue for each ad clicked on their website or app. A publisher stands to profit by attracting or generating click fraud to his/her site.

2. Dataset Description

The provided training dataset consists of 3,173,834 clicks from 3,081 publishers with status from 9/2/2012 to 11/2/2012. The validation dataset consists of 2,689,005 clicks from 3,064 publishers with status from 23/2/2012 to 25/2/2012. The test dataset consists of 2,598,815 clicks from 3,000 publishers with status from 8/3/2012 to 11/3/2012. The dataset characteristics are given in Table 1. In specific, there are two separate files for this data competition: publisher database and click database, both provided in CSV format.

Table 1: The dataset characteristic

Dataset	# of Publisher	# of Total Click	Time Window
Training	3,081	3,173,834	9/2/2012 - 11/2/2012
Validation	3,081	2,689,005	23/2/2012 - 25/2/2012
Test	3,000	2,598,815	8/3/2012 - 11/3/2012

The publisher database records the publishers profile. The example of publisher data is shown in Table 2. Noted that, only training data have status, the status of validation and test data are withheld by the organizers.

Table 2: Data for publisher's details

partnerid	bankaccount	address	status
8jcuw	?	2bw2ihs0ygkkwog4	OK
8jcru	5498xi9fdu040ogk	325gub2jzocgks84	OK
8jciv	?	48v6lbinzwaocsok	OK

where partnerid is unique identifier of a publisher; Bankaccount is Bank account associated with a publisher (may be empty); address is mailing address of a publisher (obfuscated; may be empty). Status is label of a publisher, which can be "*OK*", "*Observation*" or "*Fraud*". The label of publishers is identified by experts.

On the other hand, the click database records the click traffics. The example data for the click log is shown in Table 3.

Table 3: Data for publisher's click log

id	iplong	agent	partnerid	cid	cntr	timeat	category	referer
13417861	2885180193	MAUI	8jkh1	8gp8w	vn	2012-02-09 00:00:00.0	co	?
13417871	702430144	MAUI	8ihpi	8ffid	tz	2012-02-09 00:00:00.0	mc	?

where id is unique identifier of a particular click; iplong is public IP address of a clicker/visitor; agent is phone model used by a clicker/visitor; cid is unique identifier of a given advertisement campaign; cntr is Country from which the surfer is; timeat is Timestamp of a given click (in YYYY-MM-DD format) category is Publisher's channel type; referer is URL where the ad banners were clicked (obfuscated; may be empty).

The goal of this competition is to build a data-driven methodology for effective detection of fraudulent publishers. In particular, the task is to detect "Fraud" publishers (positive cases) and separate them from "OK" and "Observation" publishers (negative cases), based on their click traffic and account profiles.

3. Feature Extraction

Click fraud can be generated using a variety of approaches (Dave et al., 2012), such as (1) botnets (where malware on the users computer clicks on ads in the background),(2) tricking or confusing users into clicking ads (e.g., on parked domains), and (3) directly paying users to click on ads. In dealing with various fraud pattern, we first need to extract publisher's feature from various statistics such as mean, standard deviation, count from different views and different time granularity of the publisher. With these feature, we can choose the most discriminative feature to build the effective classifier.

3.1 Publisher's Statistics

We calculate the basic statistics of the publisher, unique count of attribute such as iplog, country, agent, referrer, cid and Total visit. The features and description are shown as follows:

Table 4: Publisher's statistics

Feature	**Description**
unique_count(iplog)	unique count of the IP
unique_count(country)	unique count of the country
unique_count(agent)	unique count of the agent
unique_count(referrer)	unique count of the referrer
unique_count(cid)	unique count of the cid
total_visit	count of the click log's row

3.2 Access Statistics by IP

The fraud users may visit the advertisements from the same ip address. In order to capture this, we calculate the average access, standard deviation, counting by grouping each IP for each publisher in different time granularity (by second, by min, by day). For example, in the Table 5, suppose it is the full click log for the publisher "$8kxij$", we can get the average access by IP in minute granularity is 5, standard deviation is 0, and the counting for the ip $2,919,155,822$ visit is 5. We can also get the same statistics in different time granularity such as by day, hour, second. The feature created is shown in Table 6.

Table 5: Example of fraud pattern from same IP

partnerid	iplong	agent	cid	cntr	timeat	category	referer
8kxij	2,919,155,822	Nokia X6	8gava	us	2012-02-09 07:23:13.0	mc	?
8kxij	2,919,155,822	Nokia X6	8gava	us	2012-02-09 07:23:13.0	mc	?
8kxij	2,919,155,822	Nokia X6	8gghw	us	2012-02-09 07:23:13.0	mc	?
8kxij	2,919,155,822	Nokia X6	8gghw	us	2012-02-09 07:23:19.0	mc	?
8kxij	2,919,155,822	Nokia X6	8gava	us	2012-02-09 07:23:19.0	mc	?

We also believe that there are patterns that the fraud users will visit the advertisements not from the same ip address but from the same subnetwork(See Table 7). Regarding this, we try to obtain the same statistics by each subnetwork instead of each ip. The subnetwork of different granularity can be obtained by dividing the IP by 1,000 or 1,000,000

Table 6: Access Statistics by IP

Feature	Description
avg_ip_sec	average visit by ip in second level
std_ip_sec	standard deviation of average visit by ip in second level
count_ip_sec	sum of visit count (larger that 2) by ip in second level
avg_ip_min	average visit by ip in minute level
std_ip_min	standard deviation of average visit by ip in minute level
count_ip_min	sum of visit count (larger that 2) by ip in minute level
avg_ip_hour	average visit by ip in hour level
std_ip_hour	standard deviation of average visit by ip in hour level
count_ip_hour	sum of visit count (larger that 2) by ip in hour level
avg_ip_day	average visit by ip in day level
std_ip_day	standard deviation of average visit by ip in day level
count_ip_day	sum of visit count (larger that 2) by ip in day level

and rounding the result. Based on this we can get the same statistics as shown in Table 6 in different subnetwork.

Table 7: Example of fraud pattern from same sub network

partnerid	iplong	agent	cid	cntr	timeat	category	referer
8jk0d	1,917,853,114	MSIE_6.0	8gp6q	cn	2012-02-09 11:52:27.0	se	?
8jk0d	1,917,853,057	MSIE_6.0	8gp6q	cn	2012-02-09 11:56:13.0	se	?
8jk0d	1,917,853,952	MSIE_6.0	8gp6q	cn	2012-02-09 11:58:49.0	se	?
8jk0d	1,917,853,022	MSIE_6.0	8gp6q	cn	2012-02-09 12:06:55.0	se	?

3.3 Access Statistics by Agent

Sometimes, the malicious publisher will use the same agent but different IP to access the web at different time. For example in Table 8, the fraud user using same agent MSIE 6.0 visit the same publisher using different IP at different time, while there is no other agent visit this publisher during these time period. In order to capture this behavior, we sort the click by timeat first and then agent. After the sorting, for each click log row, we can compare with the next click log row. If the agent is the same, we kept the current row, otherwise removed. With this, we can calculate the portion of the filtering rows over the total rows for each publisher. We called this feature sequence. This feature is named as $agent1$. For example, in the Table 8, after filtering, we have 8 rows left, so the final result will be 8/11. Besides that, we also try to sort the click data by agent only and calculate the statistics again as discussed above. This feature is named as $agent2$.

3.4 Access Statistics by Cid

Considering the example in Table 9, this malicious publisher access the same advertisement campaign, which cid is "8$gkwy$" many times using same IP address and agent. In this case, the clicks are likely click fraud. With this, we calculate the average access, standard deviation, counting by grouping cid for each publisher in different time granularity (by second, by min, by day). We can get the statistics like Table 6 by cid instead of by ip.

Table 8: Example of Fraud pattern using same agent but different IP at different time

partnerid	iplong	agent	cid	cntr	timeat	category	referer
8jk0d	1,917,852,952	MSIE_6.0	8gp6q	cn	2012-02-11 02:55:50.0	se	?
8jk0d	1,917,853,022	MSIE_6.0	8gp6q	cn	2012-02-11 02:56:36.0	se	?
8jk0d	1,917,853,060	MSIE_6.0	8gp6q	cn	2012-02-11 03:53:12.0	se	?
8jk0d	1,917,852,993	MSIE_6.0	8gp6q	cn	2012-02-11 04:49:42.0	se	?
8jk0d	701,380,683	Nokia2600c	8k7xb	ng	2012-02-11 04:51:58.0	se	?
8jk0d	1,917,852,993	MSIE_6.0	8gp6q	cn	2012-02-11 05:33:51.0	se	?
8jk0d	1,917,853,114	MSIE_6.0	8gp6q	cn	2012-02-11 06:30:02.0	se	?
8jk0d	1,917,853,146	MSIE_6.0	8gp6q	cn	2012-02-11 07:09:23.0	se	?
8jk0d	1,917,853,146	MSIE_6.0	8gp6q	cn	2012-02-11 07:31:21.0	se	?
8jk0d	1,917,852,993	MSIE_6.0	8gp6q	cn	2012-02-11 07:53:16.0	se	?
8jk0d	1,917,852,952	MSIE_6.0	8gp6q	cn	2012-02-11 07:55:14.0	se	?

Table 9: Example of Fraud pattern - clicking on same cid many times

partnerid	iplong	agent	cid	cntr	timeat	category	referer
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:36:05.0	co	?
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:36:16.0	co	?
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:36:40.0	co	?
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:38:01.0	co	?
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:38:09.0	co	?
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:38:26.0	co	?
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:38:50.0	co	?
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:39:16.0	co	?
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:39:27.0	co	?
8kv5w	3,251,257,947	Iphone	8gkwx	dk	2012-02-09 02:39:46.0	co	?

3.5 Access Statistics by Agent+IP

As for the IP addresses, they are actually public addresses, which may be assigned to different clickers at different time period. So an IP address may not uniquely identify a particular clicker. A better estimate is to use (IP address + agent) to identify a clicker. Based on this, we calculate the average access, standard deviation, counting by grouping IP+Agent for each publisher in different time granularity (by second, by min, by day). Again, we can obtain the statistics like Table 6 by agent+ip instead of by ip.

4. Feature Elimination

After the feature creation, we do the Backward Feature Elimination (Guyon and Elisseeff, 2003). We iteratively select the subsets of feature as follows: The first iteration of the loop is executed with all features. In the next n - 1 iterations each of the input columns of features - except the status column, is left out once. Then the end node will discard the column that influenced the prediction result the least. Then $n-2$ iterations follow where each of the remaining columns is left out once, and so on. The total number of iterations is therefore $n*(n+1)/2-1$. The decision tree is used as base classifier for training,

and average precision is used as measurement for the feature elimination. Finally, we will have all computed levels of the feature elimination together with the average precision. We specify an error threshold and select the level with fewest features that has a prediction error below the threshold. The feature after elimination is shown in Table 10.

Table 10: Features after elimination

Feature	Description
unique_count(referrer)	unique count of referrer
unique_count(cid)	unique count of cid
unique_count(country)	unique count of country
total_visit	count of the click log's row
count_ip_hour	sum of visit count (> 2) by ip in hour level
count_ip_ag_sec	sum of visit count (> 2) by ip+agent in second level
count_ip_ag_day	sum of visit count (> 2) by ip+agent in day level
count_sip2_sec	sum of visit count (> 2) by subnetwork (divided by 1,000,000) in second level
count_sip2_min	sum of visit count (> 2) by subnetwork (divided by 1,000,000) in minute level
count_sip2_hour	sum of visit count (> 2) by subnetwork (divided by 1,000,000) in hour level
count_sip_day	sum of visit count (> 2) by subnetwork (divided by 1,000) in day level
avg_sip2_day	average visit by subnetwork (divided by 1,000,000) in day level
avg_ip_ag_min	average visit by ip + agent in minute level
avg_ip_ag_day	average visit by ip + agent in day level
avg_cid_min	average visit by cid in minute level
agent1	the statistics for click data sorted by time and agent as discussed in Section 3.3
agent2	the statistics for click data sorted by agent as discussed in Section 3.3

5. Experimental Configuration

5.1 Validation Set

A validation set is usually needed for parameter adjustment to avoid overfitting. For the sake of efficiency, rather than perform full cross-validation, we simply chose an individual validation set to serve this purpose. A validation set was provided but without the label. An internal validation set was thus created from the training data utilizing the same method as that for creating the test dataset as described by the FDMA website. We do the stratified sampling on status. The publisher were randomly selected from all the publisher.

5.2 Evaluation

The evaluation of the performance on validation is using average precision criteria (Zhu, 2004), which is the same as the competition websites [1]. Given k publishers in a list ordered by their prediction score, the average precision (AP) is computed as follows:

$$AP = \frac{1}{m} \sum_{i=1}^{k} Precision(i)$$

where $Precision(i)$ denotes the precision at cutoff i in the publisher list, i.e., the fraction of correct fraud prediction up to the position i, and m is the number of actual fraud

1. http://palanteer.sis.smu.edu.sg/fdma2012/

publishers. Note that, when the i th prediction is incorrect, $Precision(i) = 0$. In our experimental setting, we set k=260 for evaluation.

5.3 Training of models

Our solution consists of a blend of many single predictors. The standard way of training a single predictor consists of two steps. In the first step, the validation set is created from the training dataset as discussed in section 5.1 and the model is trained on the remaining dataset. Then predictions for the validation set are stored. In the second step training is done on all available data with the same meta parameters as in the first step, such as number of hidden layer or number of hidden neurons per layer in neural network and so on. Last, the predictions for the test set are stored. A summary of all step is Figure 1.

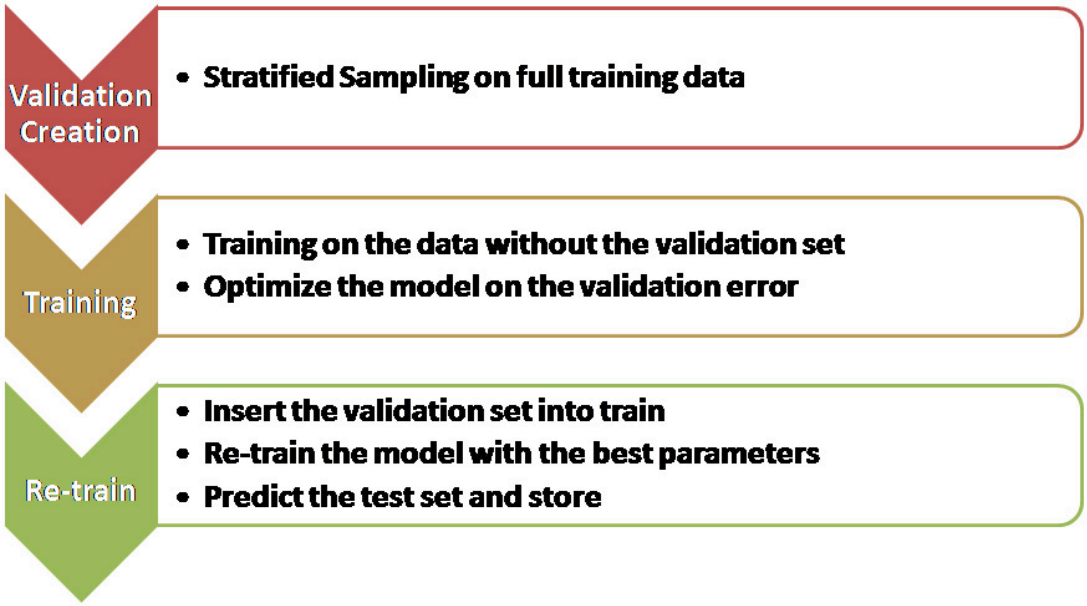

Figure 1: Process of training

6. Algorithm

6.1 FT tree

FT tree (Gama, 2004) is Tree Classifier for building 'Functional trees', which are classification trees that could have logistic regression functions at the inner nodes and/or leaves. For the parameter setting, we set min Num Instances as 15 and num Boosting iteration as 15.

6.2 RandomForest

Random forests (Breiman, 2001) are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. We set number of tree as 15 for the configuration.

6.3 REPTree

REPTree (Su and Zhang) is fast decision tree learner. It builds a decision/regression tree using information gain/variance and prunes it using reduced-error pruning (with backfitting).

6.4 LADTree

LAD Tree is (Holmes et al.) is tree models that generating a multi-class alternating decision tree using the LogitBoost strategy. It is classification technique that combines decision trees with the predictive accuracy of boosting into a set of interpretable classification rules

6.5 NBTree

NB Tree (Kohavi, a) is decision tree models which generate a decision tree with naive Bayes classifiers at the leaves.

6.6 RandomsubSpace

RandomsubSpace (Ho, 1998) constructs a classifier that maintains highest accuracy on training data and improves on generalization accuracy as it grows in complexity. The classifier consists of multiple trees constructed systematically by pseudorandomly selecting subsets of components of the feature vector, that is, trees constructed in randomly chosen subspaces. REP tree is chosen as base classifier here.

6.7 RotationForst

(**?**) is classifier that encourages simultaneously individual accuracy and diversity within the ensemble. It is achieved by creating the training data for a base classifier, the feature set is randomly split into K subsets (K is a parameter of the algorithm) and Principal Component Analysis (PCA) is applied to each subset. All principal components are retained in order to preserve the variability information in the data. Thus, K axis rotations take place to form the new features for a base classifier. We choose J48 as our base classifier here.

6.8 DecisionTable

Decision tables are a precise yet compact way to model complicated logic. DecisionTable (Kohavi, b) is rule classifier which build and use a simple decision table majority classifier.

6.9 BayesNet

Bayesian networks (BNs) provide a powerful graphical model for encoding the probabilistic relationships among a set of variables, and hence can naturally be used for classification.

BayesNet (Neapolitan, 2003) is Bayes Network learning that learn network structure, conditional probability distributions using search algorithm such as K2 based on quality measure such as simple estimator.

6.10 RPROP (resilient propagation)

RPROP (Riedmiller and Braun, 1993) is a learning algorithm for multilayer feedforward networks. To overcome the inherent disadvantages of pure gradient-descent, RPROP performs a local adaptation of the weight-updates according to the behavior of the error function. We set 5 hidden layers and 15 hidden neurons per layer for the configuration.

6.11 Tree Ensemble

Tree ensemble is classifier which learns an ensemble of decision trees. Each of the decision tree models is learned on a different set of rows (records) and/or a different set of columns (describing attributes). The output model describes an ensemble of decision tree models and is applied in the corresponding predictor node using a simply majority vote. Gini Index is used for Split criterion; Number of levels is limited to 7. Number of models is set to 500.

7. Blending

As blending algorithm we use a simple basic blending - linear blending. A linear blender is easy to implement. The most basic blending method is to compute the final prediction simply as the mean over all the predictions in the ensemble. Better results can be obtained, if the final prediction is given by a linear combination of the ensemble predictions. In this case, the combination coefficients have to be determined by optimization procedure, in general by regularized linear regression. In our case, All inputs (the predictions) are normalized to [0...+1]. We use linear blending approach and the coefficients are determined by cross-validation performance on average precision. For validation we use 10-fold cross-validation. The performance of each individual model and blending is shown in table 11.

Table 11: Performance for different algorithm

Method	Average Precision
FT Tree	36.3%
RandomForest	47.7%
REPTree	35.8%
LADTree	37%
NBTree	37.9%
RandomsubSpace	38.9%
RotationForest	42.9%
BayesNet	33.7%
RPROP	48.3%
TreeEnsemble	49.3%
Blending	52.3%

8. Conclusion

We described our approach to FDMA. In summary, we try extract the feature using basic statistics such as count, mean and statistics from different perspectives such as different time granularity, different network granularity or different attribute grouping. With the feature extraction, we do the backward feature elimination to keep the discriminative feature for building the effective classifiers. Many single predictor is trained. A linear blending of single predictor is used for final prediction. The experiments presented in this paper, and the ranking on the FDMA leaderboard[2] , suggests that our methods are effective in click fraud detection tasks.

References

Leo Breiman. Random forests. *Mach. Learn.*, 2001.

Vacha Dave, Saikat Guha, and Yin Zhang. Measuring and Fingerprinting Click-Spam in Ad Networks. In *SIGCOMM*, Helsinki, Finland, Aug 2012.

João Gama. Functional trees. *Mach. Learn.*, 2004.

Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3, March 2003.

Tin Kam Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8), August 1998.

Geoffrey Holmes, Bernhard Pfahringer, Richard Kirkby, Eibe Frank, and Mark Hall. Multiclass alternating decision trees. In *ECML '02*.

Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: a decision-tree hybrid. In *AAAI Press'1996*, a.

Ron Kohavi. The power of decision tables. In *ECML'95*, b.

Richard E. Neapolitan. *Learning Bayesian Networks*. Prentice-Hall, Inc., 2003.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *IEEE INTERNATIONAL CONFERENCE ON NEURAL NETWORKS*, pages 586–591, 1993.

Jiang Su and Harry Zhang. A fast decision tree learning algorithm. In *AAAI'06*.

M Zhu. Recall, precision and average precision. 2004.

2. http://palanteer.sis.smu.edu.sg/fdma2012/