Click Through Rate Prediction using Online, Follow the Regularized Leader and Ensemble Learning.

Hithesh Reddivari
Dept. of Electrical Engineering
reddivar@usc.edu

Harsh Singh
Dept. of Computer Science
harshsin@usc.edu

Jieci Liang
Dept. of Computer Science
jiecilia@usc.edu

Poojit Sharma
Dept. of Computer Science
poojitsh@usc.edu

Jaydeep Bagrecha
Dept. of Computer Science
bagrecha@usc.edu

Abstract

Paid advertisements displayed alongside search results,on various websites mainly on web constitute a major source of income for search companies and an unignorable avenue for getting information out in a quick and effective way for the online advertsing industry. Optimizations leading to more clicks on ads are a target goal shared by advertisers and search engines. In this context, one of the more accepted parameter to judge an ads quality is the probability of it being clicked assuming it was noticed by the user (click-through rate, CTR).In this paper we build upon a large data set with real ad clicks and impressions, acquired thanks to Avazu. We have mainly implemented an online algorithm called Follow the Regularized Leader and combine serveral models using ensemble learning including SVM (trained on a sample data) considering the trade-off between performance and our limited CPU/RAM resources.

1 Introduction

Under current technology, it is very difficult to fully quantify the emotional reaction to the site and the effect of that site on the firm's brand. One piece of information that is easy to acquire, however, is the CTR(click-through rate) defined as a ratio of number of clicks to number of impressions.In online advertising campaigns CTR is one of the most informative metrics used in business activities such as performance evaluation and budget planning. Although there are many robust machine learning algorithm for prediction, we face several real world challenges like methods for assessing and visualizing performance, practical methods for providing confidence estimates for predicted probabilities, calibration methods, methods for automated management of features etc which make accurate CTR calculation a real difficult problem to crack.In the given dataset,considering the data size, the techniques of feature selection and model training are selected based on the trade off between performance and CPU/RAM resource limit. First, we describe the data set and anomalies found inside it. We then address the problem of computing CTR for existing ads using proper feature calibration and an algorithm learning an ensemble of decision rules that can be used for predicting the CTR for unseen ads and giving recommendations to improve ads quality. We found using ensemble [2] with fine tuning of parameters worked much better than individual learning algorithms. Finally we propose some ideas of how we can take this work forward towards more scalable and robust algorithms. [3]

2 Data Analysis

In any learning problem, it is essential to first analyze the data. Given the scale of the dataset, we used data analysis tools like R and libraries like Pandas, to do data exploration. When resources were scarce we subsampled data and then analyzed it. One important analysis was finding out the number of clicks per every hour.

Since the entire dataset contained categorical features, we first started analyzing the number of unique values for each feature. We present our findings in Table 1 Also, to visualize the number of clicks in every hour, we grouped the data by the hour and calculated the sum of the number of clicks in each group using Pandas. From Figure 1, we can cleary see the temporal nature of the clicks. Every 24 hours the number of clicks fluctuate in correspondance with the time of day. During night hours the number of clicks decreases as there in low internet traffic and during peak hours everyday we see a high amount of click traffic.

To aid in feature engineering, we also analyzed the number of clicks for each banner position and the number of clicks per banner area. The banner size was calculated by multiplying C15 and C16 features which are possible banner heights and banner widths. This analysis was done by subsampling 400K rows. From Figure 2, and Figure 3 we can see that certain values of those features are relevant toward predicting the label.

Table 1: Feature Analysis

Attribute	Train	Test
id	40,428,967 unique values	4577464 unique values
click	0, 1	0, 1
hour	240 unique values, min = 14102100 max = 14103023	24 unique values min = 14103100 max = 14103123
C1	1005, 1002, 1010, 1001, 1007, 1008, 1012	1005, 1010, 1012, 1002, 1007, 1001, 1008
banner_pos	0, 1, 4, 5, 2, 7, 3	0, 1, 7, 2, 4, 3
site_id	4737 unique values	2825 unique values
site_domain	7745 unique values	3366 unique values
site_category	26 unique values	22 unique values
app_id	8552 unique values	3952 unique values
app_domain	559 unique values	201 unique values
app_category	36 unique values	28 unique values
device_id	2,686,408 unique values	291,759 unique values
device_ip	6,729,486 unique values	1077199 unique values
device_model	8251 unique values	5438 unique values
device_type	1, 0, 4, 5, 2	1, 4, 0, 5
device_conn_type	2, 0, 3, 5	0, 3, 2, 5
C14	2626 unique values	1257 unique values
C15	320, 300, 216, 728, 120, 1024, 480, 768	320, 300, 216, 480, 728, 768, 1024, 120
C16	50, 250, 36, 480, 90, 20, 768, 320, 1024	50, 250, 480, 36, 320, 90, 1024, 768, 20
C17	435 unique values	240 unique values
C18	0, 3, 2, 1	3, 0, 2, 1
C19	68 unique values	47 unique values
C20	172 unique values	162 unique values
C21	60 unique values	39 unique values

3 Data Preprocessing

3.1 Dealing with Categorical Features

Vectorization is a method used to create a vector of all the unique values of a feature and then have binary values for whenever a particular value is present. This however increases the dimensionality of the dataset tremendously. For the sake of simplicity we have experimented with One-Hot Hashing and 1-to-K encoding techniques to convert categorial features into numerical features. For One-Hot Hashing, we convert each feature value to a 64-bit MurmurHash using Python's hash function. This converts all numeric and string features to numeric hash values. For 1-to-K encoding, we first build a dictionary of all the unique values in the training dataset and then represent each feature value by their index in the dictionary. For the test data, keys which do not appear in the training data are mapped to 0.

3.2 Feature Engineering

We extract the actual hour from the "hour" feature which is represented as "YYMMDDHH". We also get the day of week by dayOfWeek $= (\text{day}\%7) + 1$ to capture the weekly trend into our model. From our analysis in Section 2, we found out that C15, and C16 could be the banner height and width. Thus, we concatenate these features before the hashing step. This represents unique resolutions of the banner. We also try experimenting by discarding features like device_id, and device_ip which are features with low variance and hence they do not contribute much towards the prediction label.

3.3 Data Pre-processing for Vowpal Wabbit

For creating the data in the format readable by Vowpal Wabbit, we converted the csv file to the format expected by Vowpal Wabbit. The format includes label followed by a | and then pairs of features and their values separated by : . Also values can

be any floating point number. Features can be any string as long as it contains no whitespace, the :, or the | characters.

4 Methods Explored

4.1 SVM experiment with subsampling of data

We used scikit-learn for experimenting with dataset provided and builiding the model.

Advantages of using scikit-learn SVM

1. Easy to use and nice documentation.
2. Gives better results even with less training samples.

Disadvantage of using scikit-learn SVM

1. Not scalable for whole dataset.
2. Computation of probabilities is erroneous and so not suitable for log-loss.
3. No online learning techniques.

Here we trained the model using 100K samples at various places of the training data.Then we averaged all of them to get thw final classifier.

Steps for using scikit-learn SVM

1. Convert data into a matrix and extract output from training dataset.
2. Decode the hashed features, set parameters and train.
3. Test using the validation dataset.

4.2 Vowpal Wabbit

We also used Vowpal Wabbit, an open source, fast, out-of-core learning system library which was developed originally at Yahoo! Research and is currently at Microsoft Research. After our initial research and realization that we cannot fit the whole data into our system's memory we decided to look for some well designed and documented online learning libraries. Vowpal Wabbit is one of them. The default learning algorithm in Vowpal Wabbit is a variant of online gradient descent. The other library we looked into was Google's sofia-ml. But the documentation, explanation of how to use and support of Vowpal Wabbit is much better than sofia-ml.

Resons/Advantages for using Vowpal Wabbit

1. Vowpal Wabbit supports online learning and optimization by default. This made our work much easier by just feeding correct optimal parameters and data type to this library, and getting the model and predictions on test data.
2. Vowpal Wabbit has Feature Hashing which allowed us to hash the categorical features directly within learning and so reduced the need to preprocess data, speeded the execution and improved the accuracy.
3. Vowpal Wabbit is able to learn from terafeature[1] datasets with ease in a very speedy way as described by John Langford[4].

Before running the Vowpal Wabbit library we needed to convert the given data into the format that can be used by it for training and testing. The data pre-processing steps are described in the data preprocessing section 3.3.
Steps for using Vowpal Wabbit

1. Creation of the custom data set to feed into the Vowpal Wabbit's online learning algorithm.
2. Training by feeding data to Vowpal Wabbit either as a file argument with -d or –data and also passing required parameter values.
3. For Testing we just feed data into Vowpal Wabbit the same way as training, but we add the -t argument to tell Vowpal Wabbit to ignore the labels, and use the -i argument to specify our training model. Also we used -p flag to specific the file where predictions for our test data should go. Predictions are normalized from 0 to 1.
4. Converting the Vowpal Wabbit to csv format and calculating the log loss to submit to Kaggle.

We ran the Vowpal Wabbit with variations in the following options, The Loss function as log Loss, different values of L1 and L2 regularizer, different epoch values and different alpha values. The top 3 results obtained are discribed in the results section.

4.3 Ensemble FTRL-Proximal

Another algorithm used in our research is based on the "Follow The (Proximally) Regularized Leader" (FTRL-Proximal) algorithm [5]. FTRL-Proximal is an algorithm developed by some Google researchers, and is used by the Google company for their online CTR prediction system. The FTRL-Proximal algorithm inherits the advantages of improved accuracy from Online Gradient Descent (OGD) and sparsity from Regularized Dual Averaging (RDA). FTRL-Proximal is an online algorithm, therefore it is effective for learning at massive scale data with limited computing resources. It takes 15-20 minutes to train the model using this algorithm on our laptop which has 4 cores and 8 GB of RAM, which is far more efficient compared to other offline learning algorithms.

In order to tune the algorithm, we divided the train dataset into training part and validation part according to the date in the "hour" column. We chose the first 9 days of data as the traing data, and the last day's data as validation data. Then, we run this algorithm over several modified datasets each time with fixed parameters (alpha = 0.1, L1 = 1, L2 = 1, epoch = 1, beta = 1, D = 2^{20}), and then trained on the dataset that output the best result by tuning the hyperparameters. Due to the lack of computing resources, we are not able to tune the hyperparameters with massive choices. To compensate for that, we tried Ensemble Learning [6]. We combined the best 2 models we have trained.

1. To get more information from the dataset, we decomposed the "hour" column from "YYMMDDHH" to day of week and the real hour columns. Because the click rate may have some relationship with day of week. Probably weekdays may have less click rate than weekends, and night time may have less click rate than day time and so on. In this way, the model will be able to grab these kinds of information. Based on the experiment results, this helps bulid better models.
2. According to our data analysis part, the device_id and device_ip have rather large number of unique values compared to the other features, 6.7% and 16.8% respectively, which may not contribute much to the result. To avoid potential noises of data, we eliminated the device_id and device_ip column from dataset. We trained the model by ignoring the device_id, ignoring device_ip, ignoring both respectively. The results prove that our assumption is valid.
3. Looking deep into the dataset, we found out that the C15 and C16 columns are similar to common screen resolution metrics. After comparing the unique pairs of C15 and C16 with commonly used resolutions, all pairs occur in the common resolution set. Therefore, we made an assumption that the C15 and C16 features jointly represent the resolution of the advertisement banner. We combined the C15 and C16 as one feature, and the result of the new model has improved.

After training various models based on the above feature engineering, we tuned the hyperparameters. There are majorly 6 hyperparameters to be tuned in the models, alpha, beta, L1, L2, epoch and D. Due to the lack of powerful computing resources, we only tuned some of the parameters, the best one is alpha = 0.15, L1 = 1.0, L2 = 1.1, epoch = 2, beta = 1.1, D = 2^{22}.

To further improve the result of the model, inspired by Ensembel Learning, we combined the top 2 models generated, and it improved our results.

5 Selected Method

Among all the algorithms we have experimentd, Ensemble FTRL-Proximal has the best result on Kaggle, 0.3922415 logloss. Given a minimum of computing resources and a large scale training dataset, the Ensemble FTRL-Proximal algorithm is effecient and produces excellent prediction accuracy.

6 Results

6.1 Vowpal Wabbit

The top three results for Vowpal Wabbit:

Table 2: Vowpal Wabbit results

Rank	Score in Leaderboard	Avg Log Loss	L1	L1	epoch	Alpha	No. of bits for hashing
1	0.3962061	0.398075	0.7	0.7	20	0.4	28
2	0.3971589	0.393805	1	1	20	1	18
3	0.3988451	0.313262	1.1	0.0000005	10	0.1	30

From the results we can observe that the best value for L1 and L2 both is 1 which may indicate the test set variations are pretty close to the train set data. Also the greater epoch value, better the result but after 20 epoch the score started dropping.

6.2 Scikit-learn SVM

The results for using Scikit-learn SVM show that it gives average results and is not suitable enough for large datasets.

Table 3: Scikit-learn SVM results

Rank	Score in Leaderboard	Avg Log Loss	C	γ
1	0.4405273	0.412353	1	0
2	0.4498723	0.422342	2	0

6.3 FTRL-Proximal

Table 4: FTRL-Proximal results

Rank	Score	L1	L2	epoch	Alpha	Beta	D	Feature Engineering
1	0.3922415	0.0005	0.00005	4	0.1	1	30	yes
2	0.3953251	1	0.1	1	0.1	1	20	yes
3	0.3960119	1	0.1	1	0.1	1	20	no

The results above is the best results we get for each major training models. Feature engineering improves the result, and reducing L1 and L2 regularizer will further improve the accuracy of prediction.

7 Conclusion

Our ultimate goal is to construct a powerful prediction algorithm for the CTR that can also be used in a recommendation system for improving ad quality. In this paper, we first described brieflythe data set, then discussed the model of estimating CTR and later used it for estimating the CTR on given test data. Based on experimental results obtained by evaluating variety of models mentioned above we found that using ensemble learning methods with fine tuning of parameters worked much better than individual learning algorithms. In the future, we plan to use non-linear methods like neural Networks and random forest which are known to give much better results in prediction task using computing resources from Amazon Web Services (AWS).

8 Acknowledgments

We would like to acknowledge the immense public help provided through Kaggle Forums. [7]

References

[1] Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. *CoRR*, abs/1110.4198, 2011.

[2] K.E. Bauman, A.N. Kornetova, V.A. Topinskii, and D.A. Khakimova. Optimization of click-through rate prediction in the yandex search engine. *Automatic Documentation and Mathematical Linguistics*, 47(2):52–58, 2013.

[3] K. Dembczynski, W.Kotlowski, and D.Weiss. Predicting ads click-through rate with decision rules. In *WWW 08*, 2008.

[4] John Langford. Vowpal wabbit for fast learning. `http://blogs.technet.com/b/machinelearning/archive/2014/08/13/vowpal-wabbit-for-fast-learning.aspx`, August 2014.

[5] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad click prediction: a view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2013.

[6] David M.J. Tax, Martijn van Breukelen, Robert P.W. Duin, and Josef Kittler. Combining multiple classifiers by averaging or by multiplying? *Pattern Recognition*, 33(9):1475 – 1485, 2000.

[7] tinrtgu. Beat the benchmark with less than 1mb of memory. `http://www.kaggle.com/c/avazu-ctr-prediction/forums/t/10927/beat-the-benchmark-with-less-than-1mb-of-memory`, 2014.

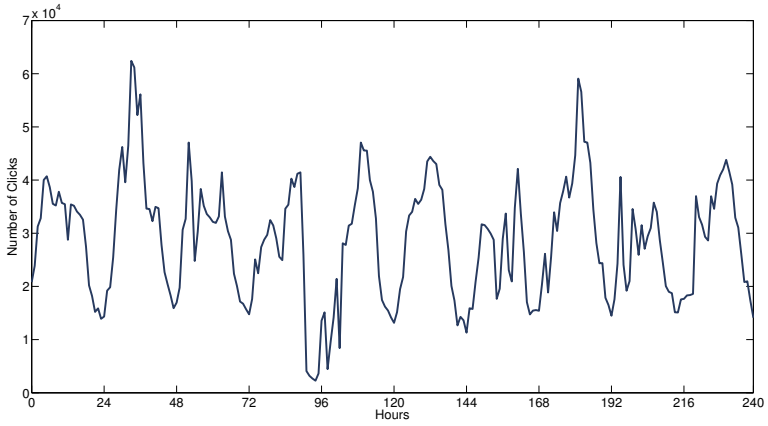

Figure 1: Clicks per Hour

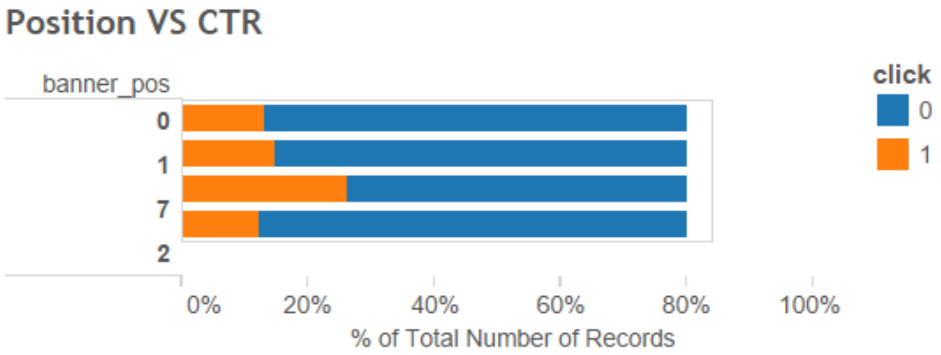

Figure 2: Position vs Click through rate

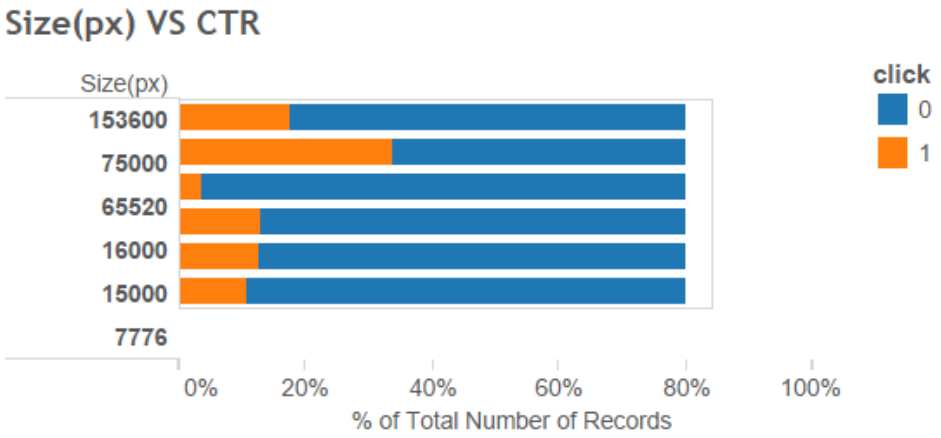

Figure 3: Size vs Click through rate