AN EMPIRICAL ANALYSIS OF THE STATE OF THE ART IN USER RESPONSE PREDICTION

Ahmet Bulut; İstanbul Şehir University, Data Science Laboratory
Beyzanur Koçak; İstanbul Şehir University, Data Science Laboratory
Süha Kağan Köse; İstanbul Şehir University, Data Science Laboratory
Oğuzhan Sağoğlu; İstanbul Şehir University, Data Science Laboratory
Hacer Tilbeç; İstanbul Şehir University, Data Science Laboratory
Contact: ahmetbulut@sehir.edu.tr

Keywords: online advertising, user response prediction, feature hashing, logistic regression

1. ABSTRACT

For effective campaign management, advertisers have to know the click-through rate of their online advertisements attracting user clicks, and the rate of users converting post clicks. Both clicks and conversions are treated as user response, and the target response depends on the goal of the marketing campaign. If the aim of the company is brand recognition, the target response should be user clicks because the number of clicks is correlated with the brand's recognition. On the other hand, if the company seeks direct sales and profit, their target response should be user conversions. User conversion is the act of purchasing a product or subscribing to the newsletter after clicking the advertisement.

In order to estimate user response, one can learn a predictive model with machine learning on historical response data. The historical data contains numerical features, such as the time of the day when the ad was shown, and categorical features, such as the category of the ad, and the gender of the user, who saw the ad.

In this study, we benchmarked two state of the art predictive models of user response. These two methods differ in the way they vectorize the input set of features. The first method proposed by He et al. uses Gradient Boosting Trees (GBT) for exploring relationships between different features and for reducing the dimensionality of the raw feature space [1]. The second method proposed by Chapelle et al. uses Feature Hashing (FH) for constructing feature vectors. In FH, each categorical feature value is first hashed to an index in the final feature vector to build, and then the corresponding value at that index is incremented by one (initially set to zero) [2].

We measured performance on a publicly available dataset (for the reproducibility of the results we reported) that contains historical user response data. We used logloss and auROC as our evaluation metrics for comparison. We observed that FH combined with Logistic Regression is scalable and performs up to par compared to its more complex GBT-based alternative.

2. INTRODUCTION

Online marketing spend has grown rapidly in recent years. According to eMarketer and Internet Advertising Bureau (IAB), the total Internet advertising spend was $260 million in 1996, and it increased to around $180 billion in 2016 [6, 7]. With close to a quarter trillion dollars spent per year, there is a growing interest in both academia and industry for solving the important challenges faced in digital marketing.

One such important challenge is the accurate prediction of user's response to an online ad. For this problem, we have historical data that captured the response behavior of users. A typical user response dataset contains attributes that belong to users and advertisements. Each such attribute represents an identifying characteristic of a user or an advertisement. Furthermore, the user's actual response to the ad also known as the label is captured as a separate attribute. The type of each attribute can either be numeric or categorical. For example, user's gender information is categorical and its value can be "male" or "female". Similarly, the device type is another categorical attribute, and its value can be "mobile", "computer", or "tablet". The total number of past clicks of a given user is a real valued numeric attribute for that user.

Using this dataset, we can build a machine learning model and use it to predict user response. One caveat however is that machine learning algorithms require each feature value to be numeric. Therefore, each categorical value has to be converted to a real value. For this purpose, one-hot encoding scheme is used. In one-hot encoding, the device type is encoded as [1,0,0], [0,1,0], or [0,0,1] for “mobile”, “computer”, or “tablet” respectively where there is a unique binary valued vector representation for each possible device type. One problem with this encoding is that since a categorical feature can have a large number of distinct values, one-hot encoded feature vectors can get very large and sparse. On large and sparse feature vectors, the learning algorithm requires more time and more data for convergence.

The above scalability issue holds true for the GBT-based state of the art method, which uses one-hot encoding for categorical features. However, one can use feature hashing for encoding categorical features more efficiently as in the second state of the art method. This distinction forms the crux of our analysis. In order to quantify and/or qualify the pros and cons of each alternative, we compared their field performance with respect to scalability and with respect to accuracy in predicting response on a publicly available user response dataset.

2.1. Organization of the paper

In Section 3, we present the details of the state of the art methods under our microscope. Later on in Section 4, we describe our experimental setup. In Section 5, we present the empirical results we obtained during our benchmark, and provide an analysis of the results along with our interpretation of them. Finally in Section 6, we conclude and point directions for future work.

3. STATE OF THE ART

In this section, we describe two state of the art methods proposed for predicting user response.

3.1. Gradient Boosting Tree (GBT) + Logistic Regression (LR): GBT + LR

The model is proposed to predict click-through rate of ads on Facebook. This hybrid model combines decision trees with logistic regression. Boosted decision trees are used for feature transformation and logistic regression is used as the prediction model.

In the model, leafs of each decision tree is considered as a categorical feature vector. If an instance (training or testing instance) ends up in any leaf of a given tree, the feature value that corresponds to the particular terminal leaf node is encoded as 1 and the rest of the feature vector is set to zero. The model captures the non-linearity of the input features by feature binning and through the conjunction of multiple features.

Figure 1 represents an example model structure with two decision trees. Input features are first transformed by decision trees, and then are fed into a linear classifier downstream. The first decision tree has 3 leaf nodes and the second decision tree has 2 leaf nodes. If an instance ends up in the first leaf of the first tree and the first leaf of the second tree, then the transformed feature vector for the linear classifier will be [1,0,0,1,0].

The computational cost of building this model is high, and it stands as a significant drawback. The authors stated that with a single core CPU, building decision trees could take more than 24 hours depending on the number of training instances, the number of trees, and the number of leaves in each tree. We tried to build a model that consists of trees in the order of hundreds over millions of raw training instances, but the build task did not finish in our compute cluster consisting of 39 cores.

3.2. Feature Hashing (FH) + Logistic Regression (LR): FH + LR

The model is proposed to predict response in display advertising. It also uses logistic regression for prediction. In order to encode features, it uses feature hashing. The idea behind feature hashing is to use hash functions for dimensionality reduction.

In this method, the size S of feature vectors is determined first. The size of the vector determines the size of the hash bucket. Then, each categorical feature value is hashed to an index in the hash bucket, i.e., [0,S-1], and the feature value at that index is incremented by 1 (initially zero). For a numerical feature value, the name of the feature is hashed to a static index in the hash bucket, and the value at the corresponding index is set to the actual numeric value of the feature. By hashing the name of a numerical feature, a fixed position is assigned for each numerical feature. The pseudo code for feature hashing is given below in Algorithm 1.

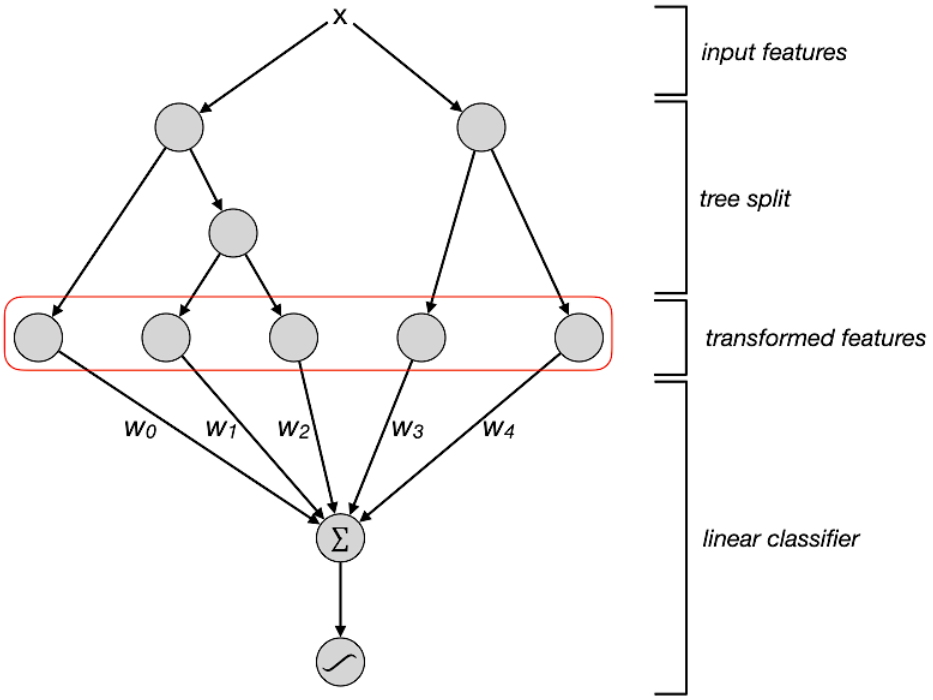

Figure 1.
The hybrid model consisting of Gradient Boosting Tree (GBT) and Logistic Regression (LR)

Require: Values for the F features, $v_1, \ldots, v_F$

Require: Hash function h, size of the hash bucket d

$x_i = 0, 1 \leq i \leq d$

for $f = 1 \ldots F$ do

if f is categorical

$i = [h(v_f) \bmod d] + 1$

$x_i = x_i + 1$

if f is numerical

$i = [h(f) \bmod d] + 1$

$x_i = v_f$

end for

return $(x_1, \ldots, x_d)$

Algorithm 1.
Feature Hashing.

Each value among F features can be of any type. In our experiments, we used Python's built-in hash function. In summary, FH is a simple mechanism to vectorize all features, and it does not require complex data processing as in GBT.

4. METHODOLOGY

In this section, we describe the dataset we used, our experimental setup, and our evaluation metrics.

4.1. Dataset

We used Criteo's publicly available display advertising dataset, which contains 45 million data rows [8]. The data corresponds to a portion of Criteo's traffic over a period of seven days. Positive (clicked) and negative (non-clicked) instances have both been subsampled. Each row of data corresponds to a display ad served by Criteo, and has 40 different values, one for each of the binary click response (1 or 0), 13 different numerical features, and 26 different categorical features. The values for the categorical features are hashed for anonymization purposes.

In our preliminary tests, we observed that using one fifth of the full dataset (roughly 9 million data rows) suffices to measure performance. The measured performance reflects the projected performance on the full dataset. We divided our dataset into two sets where 67% of it is used for training a model and the remaining 33% is used for testing the model.

4.2. Experimental Setup

For the benchmark, we used our compute cluster that contains 8 workers with a total of 39 cores and 54 GB of RAM. We used Apache Spark framework as our main distributed data processing platform. Spark provides a machine learning library, and enables us to parallelize computation on distributed data. We used Spark's MLlib library for building predictive models. The Gradient Boosting Classifier available in MLlib hogs both CPU and RAM. As an alternative, we used XGBoost library, which is designed for building boosting tree algorithms in an efficient way [3].

4.3. Evaluation Metrics

The first metric we used is the logarithmic loss (logloss), which quantifies the accuracy of a classifier by penalizing its misclassifications. Minimizing logloss is equivalent to maximizing the accuracy of a classifier. Given that a classifier assigns a probability value to each possible class and chooses the most likely class as its prediction, we can compute its performance in terms of logloss as follows:

$$\text{logloss} = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i)$$

Where N denotes the total number of test instances, y_i denotes the actual binary response for the i^{th} test instance, and $\hat{y}_i$ denotes the classifier's response prediction in [0,1] for the i^{th} test instance.

The second metric we used is the area under the Receiving Operating Characteristic curve, which is denoted as auROC. An ROC curve plots the true positive rate (sensitivity) against the false-positive rate (1 − specificity) [4]. The sensitivity, which is another term for recall, corresponds to the ratio TP / (TP + FN) while the specificity corresponds to the ratio TN / (TN + FP). In our experiments, we used Spark's areaUnderROC metric that lives in the module named *evaluation*. The score is calculated as:

$$AUROC = \int_0^1 \frac{TP}{P} d\left(\frac{FP}{N}\right)$$

An ROC curve goes from the bottom left of the graph to its top right where the graph climbs steeply on the left side for a good classifier. The closer the ROC curve is to the upper left corner, the better is the classification performance. For a given ROC curve, a high auROC value represents a better result.

5. RESULTS AND DISCUSSION

Since numerical features can be used as is without requiring any special encoding, we first tested the GBT + LR's performance on numerical features only. For GBT, we used trees of depth 7 and set the number of boosting iterations to 30. This also forms our first baseline. As shown in the top row of Table 1, this baseline has a logloss of 0.50149 and auROC of 0.72158. With categorical features included, the performance of GBT + LR improved to a smaller logloss value of 0.47709 and a higher auROC value of 0.76115. From these two sets of results we obtained with GBT + LR (the first two rows of Table 1), it is clear that using categorical features as well increase the performance of the model.

Figure 2 represents each model's ROC curve. The auROC scores of GBT + LR and FH + LR were close, but FH + LR model had the best score.

As can be seen in the bottom row of Table 1, with both numerical and categorical features included, FH + LR had a slightly worse logloss of 0.47958 and slightly better auROC of 0.76141 compared to GBT + LR.

As summarized in Table 2, the training of GBT + LR on the full dataset was infeasible due to the one-hot encoding of categorical features and it did not complete in our compute cluster. On the contrary, FH + LR does not require a computationally inefficient encoding as in GBT and is far more scalable compared to GBT + LR. On the full dataset, FH + LR took 3 hours to train and test.

In order to be able to compare both methods, we performed a separate test on the smaller dataset and encoded all features using FH for scalability. In this setting, the training time of GBT + LR was 80 minutes while the training time of FH + LR was 50 minutes as shown in Table 2.

When we compare the performance of GBT + LR against that of FH + LR, we see an interesting trade-off. Building GBT with one-hot encoded categorical features was impractical for large datasets with large feature vectors. And it offered a slight performance advantage in only one of the evaluation metrics we considered. Under the light of these results, we can state that FH + LR is a better choice since it can scale well with arbitrarily large datasets and arbitrarily large number of features.

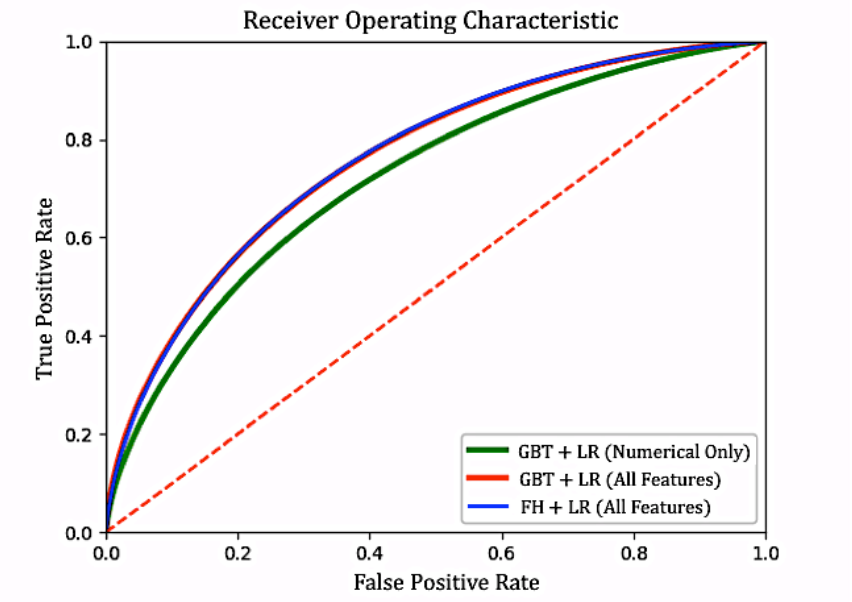

Figure 2.
The ROC curve representation of GBT + LR (numerical only), GBT + LR and FH + LR models

Features	Model	Logloss	auROC
Numerical Only	GBT + LR	0.50149	0.72158
All (Numerical + Categorical)	GBT + LR	**0.47709**	0.76115
All (Numerical + Categorical)	FH + LR	0.47958	**0.76141**

Table 1.
The performance comparison between GBT + LR vs. FH + LR with respect to logloss and auROC

Features	Model	# of instances: 9m ($1/5^{th}$ of the dataset)	# of instances: 45m (the full dataset)
All (Numerical + Categorical)	GBT + LR	80 mins	∞
All (Numerical + Categorical)	FH + LR	**50 mins**	**3 hours**

Table 2.
The training time comparison between GBT + LR vs. FH + LR with all features included

Since in practice, even the slightest performance improvement in logloss might translate into millions of dollars, one can boost the logloss performance of FH + LR in two simple ways:

i. by increasing the number of training instances since it is highly scalable, and / or
ii. by crafting new composite features such as feature conjunctions in order to capture complex relations between different features.

These simple adjustments may not be feasible for GBT + LR since it cannot scale beyond a certain point for a given hardware configuration.

The overall take-away from these results is that although GBT-based model has a slightly better logloss, we suggest the practitioners to use FH + LR since the performance gap can easily be eliminated with more training data and with more features.

6. CONCLUSIONS

We analyzed two state of the art methods (GBT + LR vs. FH + LR) proposed for predicting response. Both methods used LR as their regressor. These two methods differ in the way they encode categorical features. We observed that even though the performance of GBT + LR is slightly better in terms of logloss, we prefer using FH + LR since it is highly scalable in comparison to GBT + LR.

In the future, we plan to use deep neural networks (DNN) for estimating binary response as in [5] instead of the standard LR after the initial FH stage performed for feature encoding. We expect the new FH + DNN to have a much better logloss performance compared to FH + LR.

7. REFERENCES

[1] He, Xinran, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers and Joaquin Quiñonero Candela. *Practical Lessons from Predicting Clicks on Ads at Facebook.* Proceedings of the 8th International Workshop on Data Mining for Online Advertising, 2014.

[2] Chapelle, Olivier, Eren Manavoglu and Rómer Rosales. *Simple and Scalable Response Prediction for Display Advertising.* ACM Transactions on Intelligent Systems and Technology (TIST) 5: pp 1-34, 2014.

[3] Chen, Tianqi and Carlos Guestrin. *XGBoost: A Scalable Tree Boosting System*. In 22nd SIGKDD Conference on Knowledge Discovery and Data Mining, 2016.

[4] Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval.* Cambridge University Press, Cambridge, 2008.

[5] Zhang, Weinan, Tianming Du, Jun Wang. *Deep Learning over Multi-Field Categorical Data: A Case Study on User Response Prediction.* Proceedings of the 38th European Conference on Information Retrieval, 2016.

Internet References
(last download: July 17th, 2017):

[6] Internet Advertising Bureau Internet Advertising Revenue Report 2015 Full Year Results. Retrieved from https://www.iab.com/wp-content/uploads/2016/04/IAB-Internet-Advertising-Revenue-Report-FY-2015.pdf

[7] Digital Ad Spending to Surpass TV Next Year. Retrieved from http://www.emarketer.com/Article/Digital-Ad-Spending-Surpass-TV-Next-Year/1013671

[8] Criteo Labs. *Kaggle Display Advertising Challenge Dataset.* 2014. Retrieved from http://criteolabs.wpengine.com/downloads/2014-kaggle-display-advertising-challenge-dataset/