Ad Click-Through Rate Prediction

Jian Li
Department of Computer Science and Engineering
University of California, San Diego
jil900@eng.ucsd.edu

Xihe Zhang
Department of Electrical and Computer Engineering
University of California, San Diego
xiz512@eng.ucsd.edu

***Abstract*—In this article, we will make predictions on click-through rate, which is an important metric in online advertising. Our data set comes form Avazu click-through rate prediction competition on Kaggle. In the first section, we explore the data set in details and analyze what kind of features are probably useful for our prediction task. Then, we come up three ways to transform the data into feature set based on hash and one-hot encoder. After that, we start from the Naive Bayes method as baseline and compare the performance of different models, including some of the state-of-the-art algorithms such as FTRL-Proximal Batch algorith. The final results show that our model produce excellent performance in terms of log loss.**

***Keywords*—Click-Through Rate, FTRL, Feature Engineering**

I. Introduction

In online advertising, click-through rate (CTR) is a very important metric for evaluating ad performance. As a result, click prediction systems are essential and widely used for sponsored search and real-time bidding. With an accurate and robust prediction of CTR, advertisement companies are able to select ads that yield more profit. Together with price per click, CTR will determine the total income from the ads. Therefore, several CTR competitions are held in recent years and many efficient and accurate algorithms have been proposed. In this paper, we will discuss the CTR problem in details and build our model upon current state-of-the-art algorithms. In the following sections, we will explore the data, restate the predictive task, list related literature, introduce our model and analyse the result.

II. Data Analysis

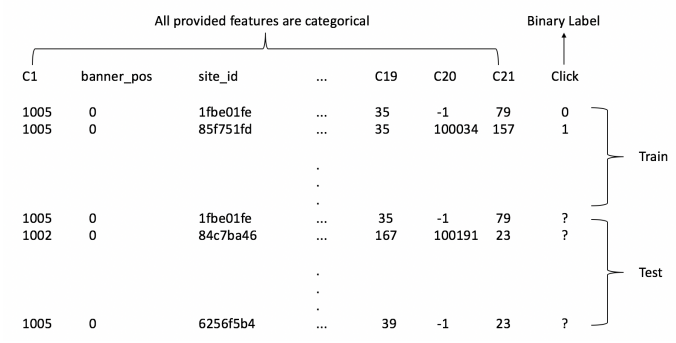


Fig. 1. Overview of the data set

Take an overview of our data set, we can find that all of the provided information are categorical, and the label of the data set is either 0 or 1, which means it was clicked or not. But the bad news is that there are some anonymous features which can not be analyzed intuitively, and some of the features describe the objective in different way. Therefore, we have to try to find the hidden relation between each feature.

TABLE I. Number of values in each feature

C1	banner_pos	site_id	site_domain	site_categ	app_id	C19
6	5	893	780	16	704	37

app_domain	app_categ	device_id	device_ip	device_model	C20
55	19	7201	40376	2473	137

device_type	dev_conn_type	C14	C15	C16	C17	C18	C21
4	4	420	5	6	128	4	29

Look at the Table I, it shows the number of values in each categorical feature. Some only have less than 10 values, we may assume this kind of feature is informative. But some other features can take a lot of values, such as device_id and device_ip, which requires us to dig more information.
Since all of the features are non-numerical, correlation or some other numerical analysis methods are useless. One way we tried to understand the feature is to draw the network relation. Take the model of device and the id of device for example, we drew first 500 pairs, the structure of the network is pretty clear.

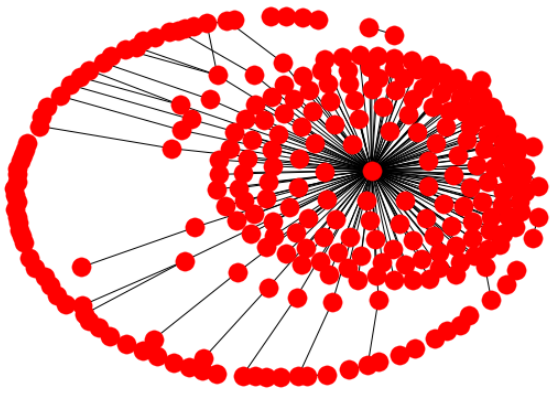


Fig. 2. Network representation of device model and device id

This kind of hierarchical structure is also obvious in our daily life, we may also identify an item from top to bottom level. For device, if we go through from the device model to id, then ip, we may consider this as an identification

process of user. It is the regular way to identify a person on Internet. If we just directly feed this three features into model, model itself may not be able to detect this kind of structure. Therefore, we may find other ways to represent this feature before sending them to the model. Not only the device has this hierarchical structure, site and app also has the same characteristics. And surprisingly, we found C14–C16 also have similar structure, but we cannot interpret it right now because they are anonymous features. This kind of finding can help us form a better representation of our features as introduced in the later section.

III. Predictive Task

Following the analysis of the data set, we would like to state our predictive task. Given a datum containing both the user and ad related information, we will make a prediction of how likely it is clicked. This is known as predicting ad click-through rate (CTR), which is the core problem to on-line advertising industry. Generally, the advertising company pays according to the number of clicks. Therefore what matters most is not the price per click, we should also take into consideration the click-through rate. In this context, a precise estimation of CTR is of great significance in industry.

During the pre-processing procedure, we split the data set by 80-20, using the last 20 percent of data as test set. The clicked information will be manually truncated from the input feature vector, while serving as the ground truth value to evaluate our predictive model. We will use hash trick and one-hot encoder to process input data and implement feature engineering approach to yield more representative features.

The predictive model is evaluated by LogLoss (logistic loss), given as

$$l_t(w_t) = -y_t log(p_t) - (1 - y_t) log(1 - p_t), \tag{1}$$

where y_t is the target label and p_t is the predicted probability of the ad being clicked. In result section, we will compare each model in terms of LogLoss.

IV. Related Literature

Our data set comes from Avazu click-through rate prediction competition on Kaggle. We read the top 5 solutions for the contest and they mainly focus on two state-of-the-art algorithms: FTRL-Proximal Batch algorithm [1][3] and Field-aware Factorization Machines[2] .

FTRL-Proximal algorithm evolved from online gradient descent (OGD), which has been proved very effective and yielding excellent prediction accuracy. Given the log loss, it is straightforward to show the gradient is $\Delta l_t(w) = (p_t - y_t)x_t$. Given a sequence of gradients, OGD performs the update as

$$w_{t+1} = w_t - \eta_t g_t \tag{2}$$

where η_t is a non-increasing learning-rate schedule. Unfortunately, OGD is not particularly effective at producing sparse models. Some more advanced algorithms such as Regularized Dual Averaging (RDA) algorithm performs well in the accuracy and sparsity tradeoff. FTRL-Proximal algorithm integrates the accuracy of OGD and the sparsity of RDA. The update rule now becomes

$$\begin{aligned} w_{i+1} &= arg\min_w (\sum_{r=1}^{i} g_r w + \frac{1}{2}\sum_{r=1}^{i}\tau_r ||w - w_r||_2^2 + \lambda_1 ||w||_1) \\ &= arg\min_w (w\sum_{r=1}^{i}(g_r - \tau_r w_r) + \frac{1}{2}||w||_2^2 \sum_{r=1}^{i}\tau_r + \lambda_1 ||w||_1 + c) \end{aligned} \tag{3}$$

where $\sum_{r=1}^{i}\tau_{rj} = \frac{\beta + \sqrt{\sum_{r=1}^{i}\tau_{rj}^2}}{\alpha} + \lambda_2,\ j = 1...n$

A practical implementation of FTRL is shown in Algorithm 1, which will be used in our model.

Algorithm 1 Per-Coordinate FTRL-Proximal with L_1 and L_2 Regularization for Logistic Regression

#With per-coordinate learning rates of Eq. (2).
Input: parameters α, β, λ_1, λ_2
$(\forall i \in \{1, \dots, d\})$, initialize $z_i = 0$ and $n_i = 0$
for $t = 1$ **to** T **do**
  Receive feature vector $\mathbf{x}_t$ and let $I = \{i \mid x_i \neq 0\}$
  For $i \in I$ compute

$$w_{t,i} = \begin{cases} 0 & \text{if } |z_i| \leq \lambda_1 \\ -\left(\frac{\beta + \sqrt{n_i}}{\alpha} + \lambda_2\right)^{-1}(z_i - \text{sgn}(z_i)\lambda_1) & \text{otherwise.} \end{cases}$$

  Predict $p_t = \sigma(\mathbf{x}_t \cdot \mathbf{w})$ using the $w_{t,i}$ computed above
  Observe label $y_t \in \{0, 1\}$
  for all $i \in I$ **do**
    $g_i = (p_t - y_t)x_i$ *#gradient of loss w.r.t. w_i*
    $\sigma_i = \frac{1}{\alpha}\left(\sqrt{n_i + g_i^2} - \sqrt{n_i}\right)$ *#equals $\frac{1}{\eta_{t,i}} - \frac{1}{\eta_{t-1,i}}$*
    $z_i \leftarrow z_i + g_i - \sigma_i w_{t,i}$
    $n_i \leftarrow n_i + g_i^2$
  end for
end for

The other algorithm: Factorization Machines(FMs) have been widely used in solving CTR problems. FMs managed to learn the latent vectors for all the features and the inner product of two latent vectors reflect the relative relationship between features.

$$\Phi_{FM}(w, x) = \sum_{j_1=1}^{n}\sum_{j_2=j_1+1}^{n}(w_{j_1}w_{j_2})x_{j_1}x_{j_2} \tag{4}$$

A variant of FM, Field-aware Factorization Machines (FFM), recently produced excellent performance on CTR competitions around the world. The FFM model does not assign exactly one latent vector for each feature in FM, it generates different latent vectors for each feature according to the field of other features.

$$\Phi_{FFM}(w, x) = \sum_{j_1=1}^{n}\sum_{j_2=j_1+1}^{n}(w_{j_1,f_2}w_{j_2,f_1})x_{j_1}x_{j_2} \tag{5}$$

Besides of this two fancy algorithms specifically for CTR problem, there are some researchers focusing on ensemble methods on similar problem, such as Wu[4] and He[5], which worth a mention.

V. Model Description

In this section, we would like to talk about the two parts of work we mainly focused on: feature engineering and classification algorithm. In the first part, we represented the raw features in three new ways to make the hidden information much more clear. And in the second part, we selected several suitable classifier to do the prediction.

A. Feature Engineering

As we can see in figure 3, there are altogether 6 kind of information associated with each entry: Device, Site, App, Time, Banner and other 9 anonymous categorical variables. All of them are non-numerical features.

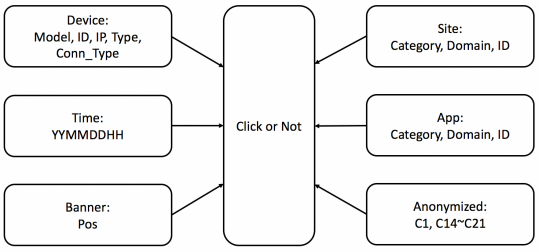
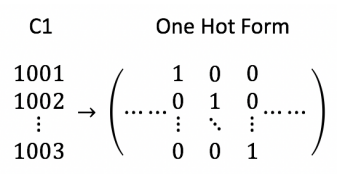


Fig. 3. Features Summary

Take the C1 for example, there are 6 different values over 80000 train samples, so that we can encode it to a matrix with size 80000×6 as shown in the picture below.



However, when the values of feature varies a lot, the matrix tends to be very large, which may bring disaster to training. Therefore, using the index of one-hot form will be a wiser process during feature generating process.

But this kind of features lacks of deeper analysis, which may be hard for the model to predict directly. In fact, every ad is clicked or ignored by a certain user, which comes an entry of data. But there is no user identification column in the raw data, we have to dig the information about user and try to define the user based on the information. From the 6 kinds of information, we can easily come up that device can be an important identifier of users. Hence, we tried to make different combinations of different attributes of devices as the feature of user. After several experiments, we found that combine the model, id and ip into one feature can improve the performance a lot. Which seems that we successfully extract the user information from device. Similarly, we can also concatenate the attributes of app and site, and get the index of one-hot form as its corresponding feature.

As shown in the figure 4, we get three new simulated identifier for user, site and app, which will replace the original corresponding features in the raw data.

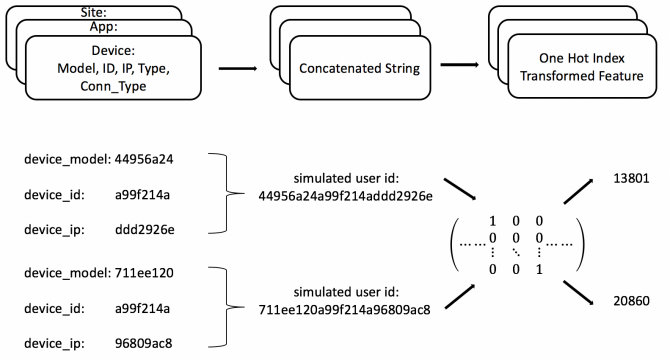


Fig. 4. Combine the features in same group, and transform the strings into one-hot index as new features

Besides, we can find that there is a lot of features uninterpretable, such as C14–C21. Some of them are related to user, some are related to ad or something else. In such kind of case, we figured out another way to process the raw feature data: no matter which group it belongs to, we just hash them according to the same criteria.

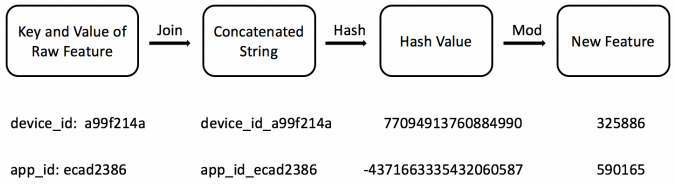


Fig. 5. Hash all the key and value pair based on same criteria

As shown in the hash figure, it is easy to understand the way we generate the new hashed feature: Concatenate the key and value pair, then hash the string and mod by a large number (avoid collision) such as 2^{20}.

So far, we talked about the three feature generators we mainly used in our model, and we will compare the performance of each kind features in Experiment Result section.

B. Classification Model

Based on the generated features, we can now apply some models on this features. The baseline of the CTR prediction task is chosen as Naive Bayes method. Because Naive Bayes model is easy to build and particularly useful for categorical features and large data sets. In our case, the probability of clicking an ad is defined as:

$$P(Y = 1|X) = \frac{P(X|Y = 1)P(Y = 1)}{P(X)} \tag{6}$$

But we have to notice that the Naive Bayes model is based on strong assumption of independent features. In fact, our click data set does have correlations between different features. Therefore, we can foresee that the performance of this simple model may not be well enough. However, this provides us with intuitive analysis of the data set and serves as baseline to evaluate other advanced model we will use. Besides, we tried KNN and Logistic Regression on the feature set for comparison. These three models are relatively simple, but their

TABLE II. PERFORMANCE OF DIFFERENT CLASSIFIERS

Model	Feature1		Feature2	
	Small Dataset	Large Dataset	Small Dataset	Large Dataset
Naive Bayes	0.725	0.619	0.547	0.484
Random Forest	0.689	0.813	1.153	1.298
KNN	0.448	0.427	0.443	0.420
GBDT	0.410	0.393	0.411	0.394
Logistic Regression	0.438	0.427	0.436	0.432
FTRL	0.393	0.367	0.382	0.359

performance turns out to be pretty good as we will state in result section.

Another good way of modeling categorical features is decision tree. But since our data set is complex, ensemble methods must help a lot. So we chose Random Forest and Gradient Boosting Decision Tree (GBDT) as two of our models to test if it works. The idea of Random Forest is sub-sampling the data set and features, while the GBDT focuses on changing the weights of weak classifiers during training. Both of them predict the results by summarizing the ideas of all the weak classifiers.

In addition to the traditional methods mentioned above, we also implemented the FTRL algorithm stated in Related Literature. This is a state-of-the-art algorithm which works best in our experiments. And this is our final model for predicting the click-through rate.

VI. EXPERIMENT RESULT

As shown in the table above, In our practical experiments, we applied different models on different features, as we talked above. As for the data set, due to the limited memory and time, we have to choose part of the original data set to train and test. Here test set also came from the original train set, so that we can test the performance of our model with labelled data. Usually, it may be called validation set during real competition. But we have to mention that we ignore the time feature, because the original data contains 400M data samples generated in several days, but we only use a small part of the data which has almost the same timestamp, so the time dimension is helpless for our prediction. Although, time maybe an important feature in real CTR prediction model, but we have to remove it from our feature set to improve the performance of our model based on current data set that we are able to process.

Firstly, we would like to show the performance of different classification algorithm. The Table II shown above is based on the features generated from one-hot encoder, where Feature1 represents direct encoder and Feature2 represents combination encoder. Meanwhile, the small dataset contains 80,000 train samples and 20,000 test samples, while the large dataset contains 800,000 train samples and 200,000 test samples. The table shows that FTRL did the best job on predicting the click-through rate, and Gradient Decision Boosting Tree (GBDT) also performs good.

In order to figure out which kind of feature representation is much better, we applied the best algorithm: FTRL on different feature set. Here Feature1 means directly encode each column into one-hot form and get its index; Feature2 means dig some information from provided data, combine them and then get one-hot index; Feature3 means hash all the features based on the same hash criteria. Figure 6 shows the logloss on train set and test set respectively over 50 iterations:

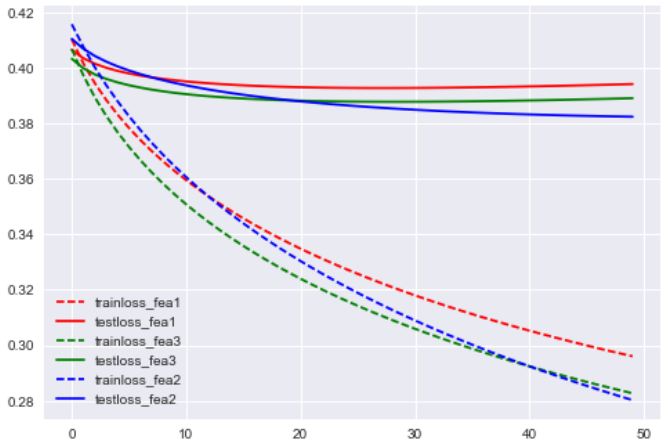


Fig. 6. Train and Test Logloss on different features with small data set

From the logloss curve, we can easily find out that the second form of feature representation performs much better than others. When it comes to 30 iterations, the model based on other 2 features start over-fitting the train data, while the model based on second feature still has a lot of space to improve the performance. Hence, we can conclude the second feature form did justify our assumptions: some features can be combined to generate identifiers. For example, the model, id and ip of device can be regarded as the user identifier. And this kind of combined feature is much more helpful for prediction.

In conclusion, we analyze the feature relations by network, and come up three ways to encode the features that not only improve the performance, but also save the memory and time. The FTRL algorithm finally achieves incredible score on the Avazu data set, which shows the robustness and power of our model.

REFERENCES

[1] McMahan, H. Brendan, et al. "Ad click prediction: a view from the trenches." Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 2013.

[2] Juan, Yuchin, et al. "Field-aware factorization machines for CTR prediction." Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 2016.

[3] McMahan, H. Brendan. "Follow-the-Regularized-Leader and Mirror Descent: Equivalence Theorems and L1 Regularization." AISTATS. 2011.

[4] "A two-stage ensemble of diverse models for advertisement ranking in KDD Cup 2012." In KDDCup. 2012.

[5] He, Xinran, et al. "Practical lessons from predicting clicks on ads at facebook." Proceedings of the Eighth International Workshop on Data Mining for Online Advertising. ACM, 2014.