Survey of Approaches for Click-through Rate Prediction

Qingjiao Li
Computational Biology
qingjial@usc.edu

Richard Li
Chemistry
liry@usc.edu

Xiaofei Wang
Biological Sciences
xiaofei@usc.edu

Beibei Xin
Biological Sciences
bxin@usc.edu

Jianqi Zhang
Biostatistics
jianqizh@usc.edu

Abstract

In this report we survey a number of approaches for click-through rate prediction. Our goal is, for a given set of features, to predict whether or not an ad will be clicked or not. Because the dataset consisted of hundreds of millions of feature values and training samples, our approaches needed to efficiently store and analyze the data. In this report we compare the log-loss and the running time of four different approaches: naive bayes, decision tree, perceptron, and a FTRL-Proximal algorithm (a regularized linear regression). In our training and testing, the FTRL-Proximal algortihm was time efficient, memory efficient, and gave the lowest log loss.

1 Introduction

Click-through rate (CTR) is the percentage of individuals viewing a web page who click on a specific advertisement that appear on the page. It is an effective way of measuring the success of an online advertisement. For advertisers, the rate helps to gauge which ads are successful and which need to be improved. For commercial website and app owners, a high CTR helps support the site through advertising dollars, since advertisers are typically charged each time their ads are clicked. Thus, prediction of click-through rate has become an important learning problem for both sides to make smart business decisions.

Many features influence the CTR of an advertisement, including the website or app the ad put in, the position the ad located, the time and date when users view the website or app, and even the device users use. From a machine learning perspective, the objective of the problem is to combine all those possible features and predict whether an advertisement will be clicked or not. For this report we will analyze a 11 days worth of data from online advertising company Avazu made available on kaggle [1]. Key challenges to arise from the dataset, which contains millions of samples and features but a relatively low amount of clicks.

Several popular algorithms deal with such big data issue, such as parallel algorithms, online learning, decision tree, neural network, stochastic algorithms. In this project, we tried naive bayes, decision tree, perceptron and FTRL-Proximal algorithm, a regularized logistic regression. Among the four, FTRL algorithm gives the best performance, with log loss of 0.394. The log loss of the other three algorithms are 0.503, 0.5, 1.1, respectively. The rest of the report will be organized as follows: in Section 2 a brief overview of the methods will be given; in Section 3 results of different methods and comparisons between methods will be given; finally, conclusions and future directions will be presented in Section 4.

2 Proposed Methods

In this section we present, in brief, the approaches used for the CTR prediction problem. All approaches were implemented using MATLAB.

2.1 Data preprocessing

Before surveying different learning methods for the CTR prediction problem, we inspected the datasets and converted them into the appropriate format. We found that there are 22 features in the provided dataset and the majority are categorical in string format. However, most of the learning methods require the features to be numerical. Since several features have a up to millions of categories(e.g. 6,729,486 for "$device\_ip$" and 6,729,486 for "$device\_id$"), we converted the feature with K categories to real-valued numbers from 1 to K rather than binary features. Because of the large size of the training and test datasets (40,428,967 and 4,577,464 for training and test, respectively), MATLAB is unable to load these files directly. We adopted a small trick that we scanned the dataset block by block (for example, 500,000 lines per block). For each block, we checked and added the new strings to the category dictionary for each feature, then converted the features into the corresponding numerical values. While converting the test dataset, we kept the category dictionary from the training dataset and added new strings that are not present in the training dataset to expand the dictionary. All the learning approaches we tried used the numerical datasets as input except Naive Bayes (which directly worked on the string format dataset). For the training dataset there were a total of 9,449,229 unique features.

2.2 Naive Bayes

In this section we explain the Naive Bayes approach. We assume we are given a feature vector $\mathbf{x_i}$ where $\mathbf{x_i} = (x_{i,1}, ..., x_{i,D})$, $D = 9449229$ and each $x_{i,d} \in \{0, 1\}$ is an indicator variable, i.e., whether, feature $x_{i,d}$ appears. We want to predict the probability $P(click|\mathbf{x_i})$. Using the Naive bayes assumption this probability is:

$$\begin{aligned} P(click|\mathbf{x_i}) &= \frac{P(\mathbf{x_i}|click)P(click)}{P(\mathbf{x_i})} && (1) \\ &= \frac{P(\mathbf{x_i}|click)P(click)}{P(\mathbf{x_i}|click)P(click) + P(\mathbf{x_i}|noclick)P(noclick)} && (2) \\ &= \left(1 + \frac{P(\mathbf{x_i}|noclick)P(noclick)}{P(\mathbf{x_i}|click)P(click)}\right)^{-1} = \left(1 + \frac{\pi_0}{\pi_1} \prod_{d:x_{i,d} \neq 0} \frac{\theta_{0,d}}{\theta_{1,d}}\right)^{-1} && (3) \end{aligned}$$

where $\pi_0 = P(noclick), \theta_{0,d} = P(x_{i,d}|noclick)$ and $\pi_1, \theta_{1,d}$ represent the analgous parameters for click.

We have already shown in class how to do parameter estimation from the data set. Naive Bayes is a parametric approach, so we simply go through the training data and count values. π_1 is the proportion of clicks in the training sample, and $\theta_{1,d}$ simply counts the number of times feature d appears in all training samples labeled 'click'. Once we have estimated the parameters from the training data, it is straightforward to predict the probabilities on the testing data. We may assign small probabilities, say 0.1, to features that are present in the testing dataset but not the training dataset.

Note that the feature defined as a series of categorical variables will be very sparse. Rather than represent it as a vector of length D, we represent it as a vector of length 22 and keep track of a dictionary of possible values. Thus in order to calculate the θ's amounts to indexing arrays, which is a relatively quick.

2.3 Decision Tree

Decision tree is a non-parametric supervised learning method used for classification. Given feature vectors $\mathbf{x}_i$, $i = 1, ..., N$ and labels $\mathbf{y}_i \in \{click, noclick\}$, a decision tree recursively partitions the space such that the samples with same labels are grouped together. The partition criterion is to choose features and thresholds that can minimize cross-entropy of data at each node.

Let the data at node m be reprinted by Q. For each candidate split $\theta = (j, t_m)$ consisting of a feature j and threshold t_m, partition the data into $Q_{left}(\theta)$ and $Q_{right}(\theta)$ subsets. To build a tree, select the $\theta^\star$ that minimizes the cross-entropy:

$$\theta^\star = argmin_\theta G(Q, \theta) = \frac{n_{left}}{N} H(Q_{left}(\theta)) + \frac{n_{right}}{N} H(Q_{right}(\theta)), \tag{4}$$

where $H(\cdot)$ is the cross-entropy function. Recurse for subsets $Q_{left}(\theta^\star)$ and $Q_{right}(\theta^\star)$ until the maximum allowable depth is reached. Once we have built the tree from training data, we can run a testing sample down the tree and see in which node the sample is located. The percentage of $y_i = click$ at that node is the probability of the testing sample being $click$. Moreover, since decision tree can create over-complex trees thus overfitting training data [3], we tuned maximum depth of the tree to accommodate to produce the best logloss.

2.4 Perceptron

We implemented a single-layer perceptron because of its ease of implementation and online learning properties. The vector of weight w is initiated to be all zeros. As each observation x is read in sequentially, the outcome is predicted as $\widehat{y} = I(w_{(t)}^T x > 0)$ using the current weight vector $w_{(t)}$. If the predicted outcome is different from the observed outcome, the weight is updated as $w_{(t+1)} = w_{(t)} + \alpha(y - \widehat{y})x$, where α is the learning rate $0 < \alpha < 1$. Otherwise, $w_{(t)} = w_{(t+1)}$. The whole dateset is read through recursively, until w converges.

2.5 FTRL-Proximal

FTRL-Proximal ("follow the (proximally) regularized leader") is essentially a space and time efficient logistic regression with regularization. More details of the algorithm can be found in [2], but for the sake of this report we hightlight a few unique aspects of this method. For this approach, a "lazy representation" of model coefficients is used to implement the regularization more efficiently. The step-size is adaptive "per-coordinate", i.e., the step-size changes with each iteration and is different for individual features.

We now outline the algorithm below (taken from [2]):

```
**Input**: Parameters $\alpha, \beta, L_1, L_2$
($\forall i \in \{1, ..., d\}$, initialize $z_i = 0$ and $n_i = 0$)
**for** *t = 1 to T* **do**
    Take feature vector $\mathbf{x}_t$ and let $I = \{i | x_i \neq 0\}$
    **for** $i \in I$ **do**
        $$w_{t,i} = \begin{cases} 0 & \text{if } |z_i| \leq L_1 \\ -\left(\frac{\beta + \sqrt(n_i)}{\alpha} + L_2\right)^{-1} (z_i - \text{sgn}(z_i)\lambda_1) & \text{otherwise.} \end{cases}$$
    **end**
    Predict $p_t = \sigma(\mathbf{x_t} \cdot w_t)$ using $w_{t,i}$ computed above
    Observe label $y_t \in \{0, 1\}$
    **for** *all* $i \in I$ **do**
        $$g_i = (p_t - y_t)x_i$$
        $$\sigma_i = \frac{1}{\alpha}\left(\sqrt{n_i + g_i^2} - \sqrt{n_i}\right)$$
        $$z_i \leftarrow z_i + g_i - \sigma_i w_{t,i}$$
        $$n_i \leftarrow n_i + g_i^2$$
    **end**
**end**
```

3 Results and Discussion

3.1 Comparison of the Four Methods

Table 1 shows the comparison of the logloss and computing time among the four methods.

Table 1: Comparison of the Four Methods

Methods	Naive Bayes	Decision Tree	Perceptron	FTRL
log loss	0.503	0.5	1.1	0.394
time	$\sim$ 1 day*	$\sim$ 2 hours	$\sim$ 20 minutes per run	$\sim$ 1 hour

*Note that when the Naive Bayes approach was implemented, the original dataset consisting of strings was used, and counts were based on string comparison. Running time would have been much shorter if the converted data had been used, but results were inferior compared to FTRL so the code was left as it was.

3.2 Naive Bayes

Using the parameters from the training set, the log loss for our naive Bayes approach was about 0.503. While this does better than having all probabilities equal to 0.5, it is still not that good. One major limitation of this approach is the presence of a large number of features with very small values. As can be seen from Equation 3, the probability is dependent on the product of ratios of $\theta_{0,d}$ and $\theta_{1,d}$. For features with such small counts the parameters we estimate may not be reliable.

3.3 Decision Tree

To have a first evaluation of how well decision tree can perform, we built a tree with depth 8 based on 5% of the training data. After applying this tree to predict outcomes for the testing data, the log-loss was 0.5017659, which is reasonable for the click problem. Based on this, we built the decision tree on the whole training data. As mentioned, depth of the tree is an important hyper parameter to tune.

Table 2: LogLoss value and CPU time for hyper-parameter tuning of Decision Tree

Depth-of-tree	10	11	13	14	15	16
Log-loss for CV	0.411780	0.409568	0.407046	0.407584	0.410228	0.415292
Running Time(Mins)	80	92	104	106	129	122

It was selected from range 10 to 16 with 5-fold cross validation. Table. 2 shows the results. When depth equals to 13, the decision tree achieves the lowest log-loss. Even though decision tree is a reasonable approach to model click problem, it still has some constraints. First, the depth of the tree indicates the number of features to use to classify training data. Once set, the model automatically omit other features, thus possibly losing information from original data. Second, decision tree can be unstable because small variations in the data might result in a completely different tree being generated.

3.4 Perceptron

It takes only about 20 mins to process the whole dataset once. However the logLoss was 1.1 after reading throught the dataset five times. Different α's were used without significant improvement.

3.5 Hyper-parameters tuning for FTLR

We used 5-cross-validation to tune the hyper-parameters (α, β, L_1 and L_2) on training dataset. We adopted grid searching for different parameter combinations, L_1 and L_2 are set in the range $10^{-4:0}$ and α, β are set as $\{0.1, 0.5, 0.8, 1, 1.2\}$. The logloss values are listed for some of parameters we tune in Table 3. The parameters with smallest average logloss value are $(\alpha = 0.8, \beta = 0.8, L_1 = 0.00001, L_2 = 0.0001)$. For a fixed learning rate, we found that smaller penalties for enforcing sparsity produce smaller logLoss value. For the fixed penalties, the larger learning rate gives better performance. However, the parameters with good performance via cross-validation are not guaranteed to make good predictions for test data. We tried different set of parameters to make a prediction for the test dataset, uploaded to Kaggle, and the returned logloss are shown in Table. 4. We found that the parameter set as $(\alpha = 0.1, \beta = 0.1, L_1 = 0.0001, L_2 = 0.0001)$ make best predictions for the test dataset. In contrast to the cross-validation, the smaller learning rate performs better than larger one.

The approximate CPU time are 4 hrs for each cross-validation, and 1 hr for learning the weights on full training dataset and 0.5 hr for predicting on the test dataset. Note that we did not need to use any tricks for memory savings. MATLAB for loops are notoriously slow, yet the running time for the FTRL-Proximal algorithm is very reasonable, considering the large size of the dataset.

4 Conclusion

In this project, we tried four methods: (1) Naive Bayes, (2) Decision Tree, (3) Perceptron, (4) FTRL, in order to solve the Click-Through Rate Prediction problem. For each method, we trained the model using the whole training dataset, and got a test logloss of 0.503, 0.5, 1.1(cross-validation), and 0.394, respectively. The training time cost are approximately 1 day for Naive Bayes, 2 hours for Decision Tree, 20 minutes for Perceptron and 1 hr for FTRL separately (Please note that different approaches were implemented by different group members and ran on different machine, therefore the time cost we provided in the report just show the time scale.). Among the four, FTRL, which is also a method used by Google for ad click prediction, performs the best. It appropriately updates the weights for features with small probability and is also extremely time efficient.

Because of time limitations, we only implemented these four algorithms. In the future, we could reducing the training time by implementing a memory hashing trick. In addition, we could try more sophisticated techniques, such as neural networks. This competition gave us valuable experience working with a real dataset.

Table 3: LogLoss value and CPU time for hyper-parameter tuning of FTLR

Alpha	**Beta**	**L1**	**L2**	**mean of logLoss**	**mean of CPU time (seconds)**
0.1	**0.1**	**0.001**	**0.001**	0.38952	2651
0.1	0.1	0.001	0.0001	0.38950	3320
0.1	0.1	0.001	0.00001	0.38946	3321
0.1	0.1	0.0001	0.001	0.38937	3335
0.1	**0.1**	**0.0001**	**0.0001**	0.38935	3326
0.1	0.1	0.0001	0.00001	0.38936	3312
0.1	0.1	0.00001	0.001	0.38934	3230
0.1	**0.1**	**0.00001**	**0.0001**	0.38934	3225
0.1	**0.1**	**0.00001**	**0.00001**	0.38934	3295
0.8	0.8	0.01	5	0.4004	2226
0.8	0.8	0.01	1	0.40036	2310
0.8	0.8	0.01	0.01	0.39632	3346
0.8	0.8	0.001	0.001	0.38468	3419
0.8	0.8	0.001	0.0001	0.38484	3341
0.8	0.8	0.001	0.00001	0.38455	3278
0.8	0.8	0.0001	0.001	0.38359	3257
0.8	**0.8**	**0.0001**	**0.0001**	0.38390	3395
0.8	0.8	0.0001	0.00001	0.38352	3301
0.8	0.8	0.00001	0.001	0.38332	3257
0.8	0.8	0.00001	0.0001	0.38329	3320
0.8	0.8	0.00001	0.00001	0.38333	2244

Table 4: LogLoss value for prediction on the test data

Alpha	**Beta**	**L1**	**L2**	**logLoss**
0.1	0.1	1	1	0.39510
0.1	0.1	0.1	0.1	0.39478
0.1	0.1	0.01	0.01	0.39474
0.1	0.1	0.001	0.001	0.39417
0.1	**0.1**	**0.0001**	**0.0001**	**0.39381**
0.1	0.1	0.00001	0.0001	0.39384
0.1	0.1	0.00001	0.00001	0.39384
0.5	0.5	0.001	0.001	0.39953
0.1	1	0.001	0.001	0.39474
0.1	1	0.0001	0.0001	0.39440
0.8	0.8	0.0001	0.0001	0.40099
0.8	1	0.0001	0.0001	0.39996
1	1	0.0001	0.0001	0.40977
1	1	0.0001	5	0.39907

References

[1] http://www.kaggle.com/c/avazu-ctr-prediction

[2] McMahan, H. et al, Ad Click Prediction: A View from the Trenches. *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*

[3] Breiman, Leo, et al. Classification and regression trees. CRC press, 1984.