Predict Click-Through-Rate Using GBDT+LR Model

Zi Gu, Yiqi Ye
School of Information
University of Michigan
zigu@umich.edu , yeyiqi@umich.edu

December 9, 2020

Abstract

Click-through-rate (CTR) as an important metric to measure the success of an online advertising system was rigorously investigated by large tech companies such as Facebook and Amazon. In this paper, we trained the data using logistic regression, random forest, GBDT, and GBDT+LR and compared the prediction results using AUC. As a result, we found that the GBDT+LR model performed the best with L1 Lasso regularization. The GBDT+LR model does not only outperform the AUC score but also leads to a faster training time. Finally, we simulated the Spark Structured Streaming environment to monitor CTR changes and model performance in real-time.

1 Introduction

In the age of Big Data, online business platforms like Facebook or Amazon have to keep track of a huge volume of data on their platform on a daily basis. Through tracking user engagement events like Advertisement Clicking Through Rate, businesses could make informed decisions to increase their ad revenue.

CTR stands for Click-through rate, which measures the ratio of clicks (how often someone clicked on your ad) to impressions (how many times your ad was viewed on the platform) for individual ads. CTR is an important metric because it helps companies understand their customers and inform them what works when trying to reach the target audience.

In this project, we aim to build ML models to predict CTR (click-through-rate) using historical data and test our model on streaming data, therefore mimicking an industrial application. We will stimulate streaming data by reading our test data from a file using PySpark API since we dont have a Kafka cluster available. This should have the same effect as reading off Kafka topics.

2 Data

For this project, we obtained the dataset from a Kaggle competition called Click-Trough Rate Prediction [Kaggle, 2014]. The dataset includes 11 days worth of company Avazus data with 404,290 rows of records, 21 columns of variables, and 1 response variable. Our response variable is the click field that contains 1:click and 0:non-click. Among all the variables, there are 9 categorical variables and 12 numerical variables. The features include information related to sites and devices, such as site_domain, site_category, app_id, device_model, device_type, and etc. Notice that there are also some variable names with the suffix C followed by numbers. Those are categorical variables that are anonymized due to confidentiality reasons and we dont know exactly what they represent even though they may provide valuable insights.

2.1 Exploratory Data Analysis

One of the variables is called the ads banner position, which has seven values. From the exploratory analysis, we can see that the most frequent banner positions are positions 0 and 1, but position 7 has the highest CTR. This indicates that even though there are a huge number of advertisements placed on position 0 or position 1, advertisements on position 7 would attract more clicks per view.

The same thing happened to the site categories variable. The top four categories of websites took 99% of clicks among all 26 website categories. However, the categories with the highest CTR did not align with those four categories.

Another variable that caught our attention is the device type. From the bar chart, we found that device 1 is the most used device when users browse the website. Since its way more frequent than any other devices in the data, there are more absolute values in clicks as compared to other device types. However, in terms of click-through-rate, we found out that device 0 actually has the highest percentage, followed by device 1.

Looking at clicks over hours of a day. We can see a bell-shaped distribution of clicks with a peak around 9 am to 1 pm. However, the CTR by hours of day fluctuates in the range of 0.16 to 0.18. As we can see, the hour for the highest click-through-rate was at 1 AM in the morning and the lowest at 4 AM.

3 Methodology

In this section, we detail the procedure and implementation of our process.

3.1 Preprocessing of Dataset

After we were done with the exploratory analysis, we started building machine learning models. Firstly, we saved our data frame as a parquet file. And then, we mapped all the columns as categorical columns since by definition in the Kaggle data description all of them are categorical variables. Secondly, we removed all the variables with unique values more than 70 since all of them are categorical variables – for example app id, which has 2241 unique values and would not make sense for us to perform one-hot encoding on. With this procedure, we filtered out 12 features and we only kept 88,655 rows of record. Finally, before we built a pipeline for training models in the later process, we used one hot encoding method to convert categorical data into dummy variables.

3.2 Logistic Regression - Baseline Model

We used logistic regression as our baseline model, to perform a binary classification task, the reason we choose logistic regression is that it is a widely used technique so as efficient. Logistic regression does not consume too many computational resources. As a baseline model, logistic regression needs only a little hyperparameter tuning and is easy to be regularized and could also provide us some basic insights on variable importance. Based on the given reason, we choose logistic regression as our baseline model before we move to other more complex algorithms. With hyperparameter tuning and training the model with l1 and l2 regularization on pyspark, we have obtained an AUC of 0.6 on the test set using l1 regularization with an alpha of 10.

3.3 Tree-based Models

After building our baseline model, we decided to move on to tree-based models. Tree-based models are popular in classification problems due to their adaptability and several advantages. Tree-Based models can be used for any type of data, whether they are numerical or categorical and they are robust with handling missing values and outliers. Besides, using a tree-based model requires no data preparation like standardization and other forms of transformation.

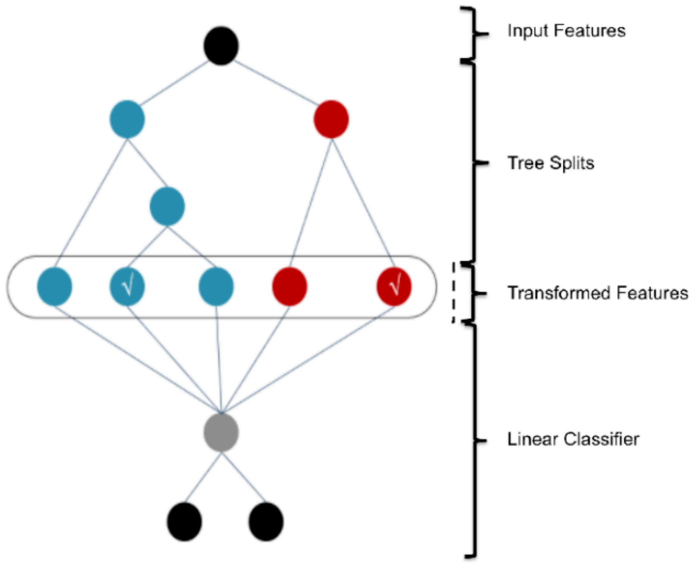
We perform two kinds of tree-based models using the ensemble method from pyspark machine learning classification packages. We implemented bagging by training random forest models and implemented boosting by training models like Gradient Boosting Classifier and Adaboost.

3.4 Random Forest and Gradient Boosting Classifier

We firstly tuned and ran a random forest classifier on the training data. The AUC result on the validation data and test data was 0.75 and 0.70 respectively. We tuned the hyperparameters of Gradient Boosting Classifier using grid search. We tuned the following hyperparameters: n_estimators, subsample, max_depth, learning_rate, and achieved a validation AUC of 0.77 and test AUC of 0.7. This gives lots of improvements from our baseline model Logistic Regression. However, it took a while for us to finish training the model.

3.5 GBDT+LR

Since running and predicting using the GBDT model would cost a longer period of time, and that in an industrial setting we often need to deal with billions of rows of data, wed prefer a faster, simpler algorithm for prediction like Logistic Regression. It would be great to have a model that trains and predicts as fast as Logistic Regression and yet contains a more powerful feature engineering to boost up the predictive performance. Luckily, we found a research paper by Xinran He and J, Pan from Facebook in 2014 [He, et al, 2014]. This research paper proposed a model called GBDT+LR, which has shown an extraordinary performance on CTR predictions. Just like its name, the GBDT+LR model essentially is a combination model of Logistic Regression and Gradient Boosting Tree. The GBDT is used to extract features from the training set as new training input data, and LR is used as a classifier for new training input data.



Specifically, there are the following steps:

1. GBDT first trains the original training data to obtain a binary classifier. During this step, grid search can also be used to find the best parameter combination.

2. Apart from the usual practice, when GBDT is trained to make predictions, the output is not the final binary probability value, but the leaf node position to which the predicted probability value of each tree in the model belongs. Record it as 1, so that new training data is constructed.

   For example, the figure above is a GBDT+LR model structure. Suppose GBDT has two weak classifiers, which are represented by blue and red parts. The number of leaf nodes of the blue weak classifier is 3, and the red weak classifier has the number of leaf nodes of 2. Meanwhile, the prediction result of 0-1 in the blue weak classifier falls on the second leaf node, and the prediction result of 0-1 in the red weak classifier also falls on the second leaf node. Then we remember the prediction result of the blue weak classifier as [0 1 0] and the prediction result of the red weak classifier as [0 1]. Taken together, the output of GBDT is the combination of these weak classifiers [0 1 0 0 1], or a sparse vector (array).

When using GBDT to construct new training data, it uses One-hot method. Since each weak classifier has only one leaf node outputs the prediction result, in a GBDT with n weak classifiers and m leaf nodes, each piece of training data will be converted to a 1*m sparse Vector. In the vector, there are n elements equal to 1, and the remaining m-n elements are all 0.

3. After the new training data is constructed, the next step is to input the label (output) data in the original training data into the Logistic Regression classifier to train the final classifier. The operation of extracting GBDT from the original data into new data would expanse the data in the scale of number of week classifier and number of leaf nodes. Thus, the data would become sparse and trigger high dimensionality issue. Therefore, in the Logistic Regression layer, regularization can be implemented to reduce the risk of overfitting. In the GBDT+LR model we trained, we used L1 Lasso regularization.

4 Experimental Results

The evaluation metrics we choose for comparing model performance are AUC with precision. Since in advertisement click prediction, most people would not click on the ads, there will be more 0s than 1s in our response variable. Therefore we need to focus more on precision here than other metrics such as accuracy or recall. A model that produces higher precision would enable businesses to place advertisements that are more likely to be clicked on and thus increase their revenue.

The results of the various experiments are summarized in Table below.

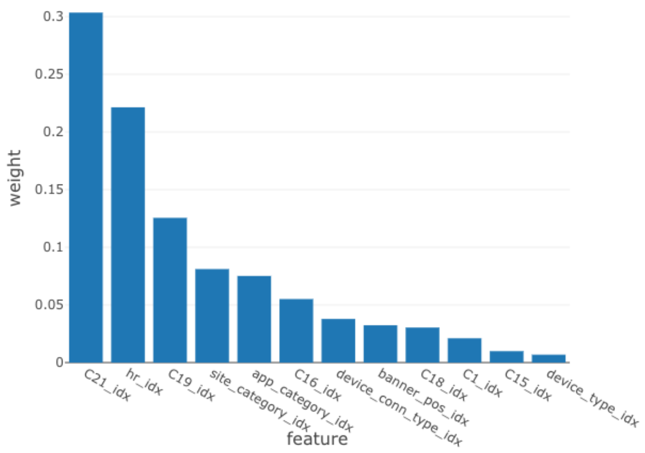
Experiment	Validation AUC	Test AUC
Logistic Regression	0.67	0.60
Random Forest Classifier	0.75	0.70
Gradient Boosting Tree Classifier	0.77	0.70
GBDT+LR (L1 Regularization)	0.74	0.71

5 Analysis of results

It seems like our final model of GBDT + LR has not improved too much from Gradient Boosting Classifier itself although it did provide some advantage over the baseline. One thing to note is that the data that is given to us contains no continuous variables and all of them are categorical features with numerous unique values even after an initial feature selection. This resulted in a huge sparse matrix after one hot encoding and led to a longer period of time of training. In addition, during our feature engineering with GBDT + LR, we also one-hot encoded the leafs of our resulting gradient boosted tree. Due to the size of our trees and number of leaves in each tree, this could further expand on our feature set and make our model harder to converge (resulting in 7000 + features). Therefore in our final logistic regression using the GBDT feature, we found out that L1 regularization performed slightly better than L2 since it automatically reduces feature space.

From the discrepancy between the validation AUC and test AUC, we found out that our GBDT + LR model is less overfitting than our GBDT model. This is probably due to the simplicity of our logistic regression + regularization despite the expanded feature space. Therefore, our GBDT + LR model is perhaps more generalizable on unseen data, aside from it being able to be trained relatively faster than just GBDT and more predictive than LR alone.

We have also examined the feature importance in our GBDT model and found out that feature C21 has the highest importance following by hour of the day, site category, etc. which gives us some interesting insight in influential factors in ad clicking (See graph below).



6 Conclusion

Our GBDT + LR model brings faster run time compared to GBDT alone and better performance than LR alone. However, as we have discussed above, there are still limitations in terms of slowing run time by sparse feature space. In the future, we could introduce some feature selection methods to get rid of less important GBDT features or incorporate Factorization Machine replacing Logistic Regression in order to achieve better run time. In order to improve our predictive performance even more, we could also try deep learning methods as it performs better generally than traditional Machine Learning methods when there are billions of rows of data.

References

He, Xinran, J. Pan, O. Jin, T. Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, R. Herbrich, S. Bowers and J. Q. Candela. Practical Lessons from Predicting Clicks on Ads at Facebook. ADKDD'14 (2014).

Kaggle, Avazu. Click-Through Rate Prediction. Kaggle, 2014. https://www.kaggle.com/c/avazu-ctr-prediction/data.