Click Through Rate Prediction

Dharamendra Kumar **Pranjay Patil** **Yash Shrivastava** **Shubhi Jain**

University of Georgia
Department of Computer Science
Athens, GA 30602
{dharamendra.kumar25, pranjaypatil, yash.shrivastav25,
shubhi.jain25}@uga.edu

Abstract

Click Through Rate (CTR) provides a numerical measure of the performance of an advertisement being displayed to the users of a website. Accurate prediction of CTR is a topic of research which involves designing models that are able to decide the advertisement to display to a user based on a feature set. This work implements some machine learning models that are able to predict the same with decent accuracy. The design of such a system involves significant amount of feature engineering to identify the features which result in the best predictions possible. The resulting system can analyze an advertisement with a given set of features and decide whether it will be relevant to the interests of the user. We present the details of our implementation of two models based on Random Forests and Wide and Deep Learning. We also present the feature engineering undertaken to identify best set of features. We conclude with a look at the accuracy of these two models on a test set.

1 Introduction

Online advertising plays an important role in supporting various websites and mobile applications. Most of the major corporation in the industry today earn a major part of their revenue from advertising. Even small companies are largely supported by ads. Click Through Rate (CTR) is the ratio of users who click on a desired link to the total number of users who view the ad [1]. In online advertising, the advertising company is usually paid only if a user clicks on the displayed advertisement. It is thus important to maximize the chances of the ad being clicked.

The motivation of this project comes from a Kaggle competition called Display Advertising Challenge of 2014, here the goal was to predict whether an ad will be clicked, based on the traffic logs. We used dataset provided by Criteo Labs which according to the challenge description was one week of data traffic.

Today, CTR estimation models are generally linear, ranging from logistic regression and Naive Bayes to FTRL logistic regression and Bayesian probit regression [4]. All of these are based on sparse features with one-hot encoding. The advantage of linear model is they are easy to implement and provides efficient model training. The disadvantage of these linear model is their low performance and inadequacy to work with non-trivial patterns. On the other side, non-linear model can work with different combinations of features and thus predict more accurately. Gradient boosting through decision trees learn how to deal with different combinational features. The problem with these models is their inefficiency to work with complex and massive data.

Deep learning has emerged as an efficient technique in computer vision, speech classification and natural language processing. Visual and textual data is said to have a spatially and temporally related, such deep structure can be used to resolve local

dependency and establish a dense representation of the feature space. With this type of learning, Deep learning can also be a possible good option to work on CTR ad prediction. In our implementation, we used both the machine learning techniques to train the model and make target predictions.

2 The Dataset

The dataset used for this project is publicly available and provided by Criteo Labs[5]. The raw data consists of two text files one each for training and testing. The training file consists of approximately 45 million records. The test file consists of approximately 6 million records.

The training file has records on each newline. Each line consists of tab separated entities. These entities represent the features for each record. Every record is a set of features and a label. There are a total of 39 features. The first 13 features are continuous features and the rest are categorical. The first value in each new record is a label. All the data is anonymized and the dataset does not provide details about what each feature represents. Although we assume that some of these features represent the information about the advertisement being displayed, some might have information about the user to whom the advertisement is being displayed.

One important thing to note about the dataset is that each feature contains large number of missing values. This was expected because many times the advertisement display might not have all the information about the user. Sometimes the feature might not be applicable to the user or the advertisement being displayed. We have taken care to fill in these missing values with appropriate filler information as described later in the report.

All this information also applies to the test dataset except that it does not have labels.

3 Implementation Details

Challenges of CTR prediction requires special treatment to machine learning techniques and good feature engineering and selection for this huge dataset. Our approach in general is to apply three machine learning algorithms for CTR prediction: TensorFlow's Wide and Deep learning, Random Forest and Gradient Boosted Decision Trees. Separate feature engineering techniques were applied for different algorithms. Finally, the accuracy was compared.

Following sections provide detailed description of each of the algorithm's implementation and how we increased the accuracy.

3.1 Wide and Deep Learning

Wide and Deep Learning is an concept introduced by Google and it can be implemented using TF.contrib.learn library. Wide and Deep Learning aims to achieve both, *memorization* and *generalization.* Memorization can be defined as learning the frequent co-occurrence of items or features and exploiting the correlation available in the historical data. Generalization, on the other hand, is based on transitivity of correlation and explores new features combinations that have never or rarely occurred in the past.

The Wide and Deep learning framework jointly train feed-forward neural networks with embedding and linear model. The Wide component include all the cross-product transformation of different categorical features available. For the Deep part of the model, a 8-dimensional embedding vector is learned for each categorical feature. All the embedding are concatenated with dense (continuous features). The concatenated vector is then fed into ReLU layers, and finally logistic output unit. The wide component and deep component are combined using a weighted sum of their output log odds as the prediction, which is fed to one common logistic loss function for joint training [2].

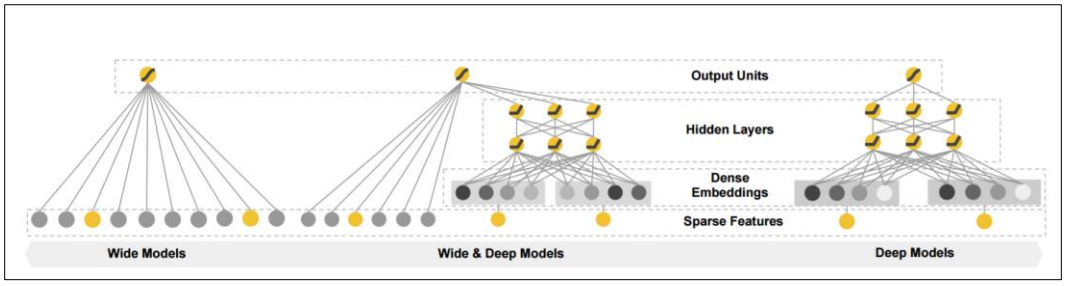


Figure 1 describes The Wide and Deep learning model. This approach works perfectly well for our problem as our dataset contains both categorical and continuous feature columns. The following sub-sections provide insight into actual implementation in TensorFlow.[3]

3.1.1 Data Preprocessing

The data was pre-processed using Pandas library available for Python. First, the tab separated file was read into Panda's dataframe in batches. The primary purpose was to fill all the missing values in the dataset. For continuous feature columns, the missing data in a particular column was filled using median and for categorical columns, mode was used. We also calculated the number of unique features in all the categorical columns which further helped in feature engineering.

3.1.1 Building the Model

The next step after preprocessing the data was to build the model using tf.contrib.learn API. An important decision to make was to decide what kind of data goes into wide and deep parts of the model. Three feature transformation techniques were used before feeding the data into the model. First, all the categorical feature column data was hashed to convert the format of data from string to numerical. The bucket size for hashing was decided according to the number of unique features in the column. Second was to generate new cross-correlated columns that took care of all the feature combinations that were rarely or never seen. The last and the most important feature transformation was to convert the categorical feature columns into embedding columns. The embedding aims to reduce the sparse feature columns into dense. The wide part of the model was trained using cross-correlated and categorical feature column, while the deep part was trained on continuous feature and embedding columns. The deep part data if fed into three hidden ReLU layer.

3.2 Random Forests Implementation

Initially when we began working on the project, we started out with implementing a Logistic Regression model that could provide us with a numerical estimate of the probability that an advertisement will be clicked by a user. The initial accuracy of logistic regression was not high and we realized that this might not be a best approach to solve this problem. We then decided to try Random Forests.

The following sub-sections describe the data processing, feature engineering, feature selection and model design of the Random Forests implementation for this work.

3.2.1 Data Preprocessing

We first used the tab separated data and created a Spark RDD. We then created a new Spark DataFrame from this RDD. The initial DataFrame had all columns set to contain string data.

3.2.2 Feature Engineering

Our feature engineering mainly revolves around the filling the missing values with appropriate data. Since we do not have information about the details contained in each of the columns, we decided that we need to go for a statistics based approach which will remain the same across all the features of the same type. That is, all missing values in the continuous features will have the same value regardless of what the column might represent. Same goes for the categorical features.

We replaced the missing values in continuous features with the median of all the values of that column. Spark allows us to calculate the median of a column in a DataFrame and also to replace the missing values with the computed median.

For categorical features we first decided to replace the missing values with the most frequent value of a column. When this implementation was executed, we realized that some columns had to many unique values and finding one value with highest frequency is consuming a lot of computing time. We thus decided to replace the missing values with the hashed representation of the word "unknown". Consider a column that might be holding the URLs of the website that the advertisement might be displayed on. This column can have a lot of unique values as even a slight change in the URL essentially makes it a new value. In this case, it does not make much sense to try to compute the one value with highest frequency of occurrence as there might be multiple values with the same frequency.

3.2.3 Feature Selection

After our initial training attempt on the large dataset using all features, we realized that the Random Forests model was not able to train efficiently. This might be due to the fact that the way Random Forests work internally might not work well with features having large number of unique values. We thus decided to use feature selection to reduce the number of columns and hence obtain a feature set that best described the data present.

For this purpose, we used the Chi Squared Selector as implemented in Apache Spark framework. We implemented the Chi Squared Selector to work only on the categorical features. The reason is the fact that categorical features are likely to have a lot of values which are repeated multiple times in a given column. But the same might not be true for the continuous features. Since Chi Squared Selector depends on the values in the columns, it makes sense that the selector be applied only to the categorical features.

The only challenge that we ran into while implementing the Chi Squared Selector is the limitation it imposes over the number of unique values in a column. The restriction imposed by Chi Squared Selector is a maximum of 10,000 unique values in a column. We had some columns which exceeded this limitation, we thus decided to skip them from feature selection but kept them in the resulting DataFrame.

3.2.4 Random Forest Model

Our Implementation of Random Forests is as follows, the model has a maximum tree depth of 8, good accuracy was obtained when we kept the maximum number of trees to 128. The impurity was set to use the gini impurity. One important thing to note is the number of max bins. This number was set to 2400000. This was again caused by the fact that a lot of values is some column were unique.

3.3 Gradient Boosted Trees

To improve the accuracy on the test set, we implemented Gradient Boosted Trees (GBTs). We used the GBT implementation of Apache Spark framework. According to the documentation of Apache Spark, GBTs are idle for binary classification and works well with both categorical and numerical features [2]. The GBT model is designed to use a max iteration of 100. The results section will discuss more about the accuracy obtained by using Gradient Boosted Trees.

4 Results

4.1 Random Forest/GBDT Classifier

To evaluate the performance and predicting the target labels(Click-1/Non-Clicked-0) the model is trained with following specification:

Machine Specification:

Specification	
Amazon EC2 Cluster	m4.2xlarge machine (1 master & 3 slaves)
Machine Memory	87.2 GB
File System	Apache Hadoop

Random Forest Hyperparameter:

Specification	
Tree Depth	16
MaxBin	2400000
Number of Tree	128

GBDT Classifier builds the model with below hypermeter:

Specification	
Max Iteration	100

4.2 Wide and Deep Tensorflow

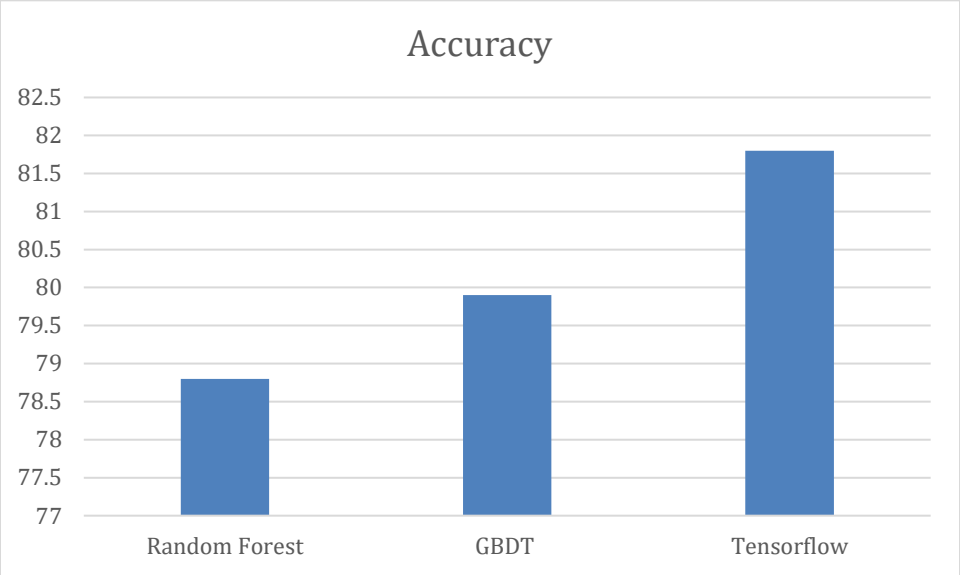
Wide and Deep Model builds on single instance with below specification:

Specification	
Amazon EC2 instance	m4.4xlarge
Machine Memory	64Gb

Following are model specification:

Specification	
Wide Columns	C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17,C18,C19,C20,C21,C22,C23,C24,C25,C26
Deep Column	I1,I2,I3,I4,I5,I6,I7,I8,I9,I10,I11,I12,I13 C1,C2,C3,C4,C5,C6,C7,C8,C9,C10,C11,C12,C13,C14,C15,C16,C17,C18,C19,C20,C21,C22,C23,C24,C25,C26
Hidden Layers	[512,256,128,64]
DNN Optimizer	AdaGrad
Linear Model Optimizer	SGD

Accuracy:



References

[1] https://en.wikipedia.org/wiki/Click-through_rate

[2]https://spark.apache.org/docs/latest/ml-classification-regression.html#gradient-boosted-trees-gbts

[3] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra,Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, Rohan Anil,Zakaria Haque, Lichan Hong, Vihan Jain, Xiaobing Liu, Hemal Shah, “Wide & Deep Learning for Recommender Systems”, June,2016

[4] Broder, A.Z.: Computational advertising. In: SODA. vol.8, pp. 992–992 (2008)

[5] http://labs.criteo.com/downloads/2014-kaggle-display-advertising-challenge-dataset/