

An Empirical Study of a Large Scale Online Recommendation System

Huazheng Fu¹(✉), Kang Chen¹, and Jianbing Ding²

¹ Guangzhou Research Institute, China Telecom Corporation Ltd., Beijing, China
{fuhuaz, chenkang}@gsta.com

² Advanced Digital Sciences Center,
Illinois at Singapore Pte. Ltd., Singapore, Singapore
jianbing.d@adsc.com.sg

Abstract. The online recommendation service has a wide range of usages for the various applications of Telecommunication companies. For such applications, the user base is usually tremendous with a variety of user characteristics and habits. Therefore, it is a challenge to achieve the high click through rate (CTR) for the online recommendations. In this paper, we proposed an approach of combining the technologies of ensemble trees and logistic regression (LR). The ensemble trees are effective in capturing the joint information of different features, which are then used by the LR scheme. In addition, to deal with the scalability issues, we implemented our system with both Apache Storm (for real-time prediction and classification) and Apache Spark (for fast off-line model training). A group of experiments were carried out with real-world data sets and the results show the efficiency and effectiveness of our proposed approach.

1 Introduction

With the rapid development of mobile Internet, intelligent terminal, cloud computing and Internet of things, Telecommunication industry has experienced a substantial increase in traffic volume and data intensity. Take China Telecom as an example, it has a bunch of systems, e.g. Customer Relationship Management (CRM), Business Intelligence (BI), Business and Operation Support System (BOSS), etc., which have recorded more than 0.75 billion pieces of user interaction information, and this number keeps increasing every day. In fact, these meta data contains very rich and valuable information related to the customers, such as the customer profile, call detail records (CDRs), web-surfing traffics, usage of data services, usages of intelligent terminals and so on. In this sense, China Telecom is surely facing the big data challenges, as known as “4V”: *volume, variety, value and velocity*.

In Telecommunication industry, the primary use of these meta data is for delicacy traffic management. The commercial values of the big data applications are reflected in two aspects: (a) have a deeper understanding of customer behaviors; and (b) help on calibrated-marketing [1].

For (a), since there is a large amount of customers' terminal information, their web surfing records of mobile devices, and other related meta data, the traditional approach is to establish the calibrated classification model for customers. With the help of deep packet inspection (DPI) technology, it is able to labeling the different groups of customers based on their online behaviors, which helps the service providers to obtain the complete customers' "portraits" and get a thorough understanding of customers' personal preferences and needs. The outcome of (a) is then utilized for advertizement and establishing the precise matching between the customers and the service packages, terminal device types and so on; and also for satisfying the customized user demands.

However, it is still a challenge how to properly use these data for delicacy traffic management. For example, when a customer logs the online business office of a Telecom company, how can we make prediction on his/her behaviors, i.e., whether or not to click on those links which are produced by the online recommendation algorithm?

This is actually a classification problem which needs real-time processing. The target is to find out what users are most likely interested in a very short time period. There are a good number of state-of-art approaches with acceptable performance proposed. Out of these, the approach which combines the ensemble trees and the probabilistic sparse linear classifier is the most suitable one for achieving our purpose.

However, all these approaches involves the model training phase, which is too time-consuming to meet the low latency requirement for the online recommendation systems. Therefore, in our design and implementation, we make the system into two separate modules, the offline data training, and the online recommendation, which can definitely run in parallel. We observe that the new features generated by the combined model earns higher AUC measure.

As the data volume is huge, in order to process as quick as possible, we choose to use Apache Spark, a Lightning-fast cluster computing engine for dealing with the offline data training module. However, due to the rigorous constraints on the response time of the online recommendation module, we choose to use Apache Storm for this module. Some other popular systems like Hadoop and Spark streaming are not taken into consideration because of their long processing delay for the inputs.

The paper is organized as follows. In Sect. 2, we begin with discussing the related work about CTR prediction. In Sect. 3, we describe the proposed algorithms. We introduce the implementation details in Sect. 4, and the experimental settings and results in Sect. 5. Section 6 concludes the work.

2 Related Work

In the machine learning research community, the CTR (click through rate) prediction problem is becoming more and more important. Recently, many people have proposed different models and methodologies for CTR prediction. For example, Neter *et al.* [2] proposed to use logistic regression, Richardson *et al.* [3]

and Graepel *et al.* [4] proposed to train the standard classifiers based on concatenation of the user and the ad features. Some other researchers suggested models which use prior knowledge like the inherent hierarchical information in log-loss models [5] or LR models [6]. In [7], Menon *et al.* used a matrix factorization approach without utilizing the user features. In [8], Yan *et al.* proposed a coupled group lasso (CGL) model to integrate the conjunction information from the user as well as the ad features. In [9], Stern *et al.* raised a probabilistic model which used user and item features together with collaborative filter information. It mapped user and item features into lower dimensions and use inner product to measure the similarity. In [10], the authors proposed a method which use boosted decision trees to transform all the features to binary values which were used for LR training. However, only use the transformed features may lose important information for the classifier. Because LR model is easy to implement, it is now becoming one of the most popular models for CTR prediction problem. However, LR is a linear model, where the contribution made by the input features to the final prediction results are independent. In consequence, it cannot capture the underlying connections among features. Better performance can be achieved by applying the ensemble trees to capture the underlying connections among features, which is then used for the LR training.

3 Algorithm Description

This section proposes a combined model structure: the concatenation of ensemble trees and a probabilistic sparse linear classifier. In the following, we will introduce how we combine the two schemes properly.

When transforming the input features to improve the accuracy of a linear classifier, there are two possible ways [10]: (a) group the continuous features into

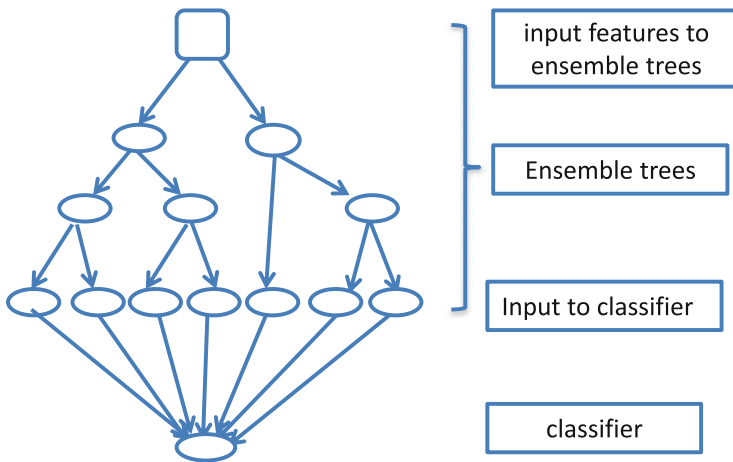


Fig. 1. The overview of the hybrid approach.

discrete bins, and (b) build tuple features. In the former case, the bin index was treated as a categorical feature from which the linear classifier can learn a piecewise constant non-linear map. In the latter case, the Cartesian product of the categorical features and the joint binning of the continuous features is calculated to build the tuple features which are later useful for the linear classifier.

As is known that the ensemble trees scheme can realize both (a) and (b) discussed above, therefore it is a powerful and convenient tool to transform features of the linear classifiers. In an ensemble tree, each individual subtree is treated as a categorical feature which is the index of the leaf node where an instance ends up. For example, consider the ensemble trees in Fig. 1. It has 2 subtrees. The left subtree has 4 leaf nodes while the right one has 3 leaf nodes. If an instance ends up at the third leaf node of the left subtree and the first leaf node of the right subtree, the result new feature produced by the ensemble trees will be a binary vector with values $[0,0,1,0,1,0,0]$. Each element of this binary vector is actually an indicator of the occurrence of the input instance to the corresponding leaf node. The ensemble trees can be realized as the gradient boosted decision trees (GBDT) [11], the random forest trees, and etc. In each learning iteration of the ensemble trees, a new tree is created and a binary vector is produced by it. We can take the ensemble trees based transformation as a supervised feature encoding whose functionality is to select those features that are more important to the classifier and jointly convert them into a compact binary-valued vector. A traversal from the root node to a leaf node represents a combination of some features. The weights of the rule set can be learnt by conducting a linear regression on these binary vectors [10]. In addition, the ensemble trees are usually trained in a batch manner.

4 System Implementation

In this section, we introduce how we implement the online classification application with the Storm stream processing engine (SPE). At first, we give a brief introduction of the Apache Storm [12] SPE. Secondly, we present the key points of implementing the online classification module on Storm. At last, we describe how the online module interacts with the offline learning module, which will be implemented with the Apache Spark [13].

4.1 The Apache Storm SPE

Storm is a distributed real-time computation system open source by twitter. In Storm, each real-time application is represented by a directed graph (called topology in Storm), of which the vertices are user-defined operators which encapsulate computation logics and the edges define the data-transmitting path, pointing from the upstream operator to the downstream operator. Note that one characteristic of Storm is that such a directed graph can be a general one, hence this are no topological constraints, e.g., it allows loops (cyclic) and each operator can have multiple upstream operators and/or multiple downstream operators, too.

There are two types of operators pre-defined by Storm. One type with special API design, called spout, is acting as the role of the data source, which mainly concerns of retrieving data from dedicated storage (file or memory) and feeding them into the application topology. The other type, called bolt, is designed to serve more general purpose. It actually can be viewed as an abstraction of any kind of computation logic implementation.

4.2 Online Classification Application on Storm

There are two major functionalities of the online classification application: (a) quick response: compute and return the classification results requested by the user in real time; and (b) online model training: take the feedback data as the input to training the model and the coefficients, and live update the corresponding classifier.

Figure 2 illustrates the topology of the proposed online classification application. It consists of two spouts (Query spout and Feedback spout) and three bolts (Feature Extractor, Trainer and Classifier). To make a reliable and scalable system, we adopt the Kafka [14] as the implementation of the message queues, i.e., Storm spouts continuously read data from the Kafka queue, and feed them into the subsequent bolts to process. Since there are two different types of input data, we have created two pairs of spout and the corresponding Kafka queue for each of them, e.g. Query spout for the user classification requests and Feedback spout for the feedback data. The spouts encapsulate every input data message to be a Tuple, which will then be sent to the Feature Extractor Bolt.

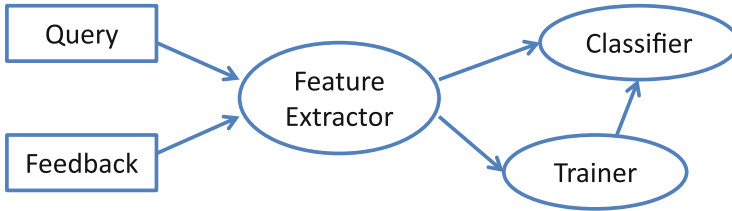


Fig. 2. Illustration of the topology of the proposed online classification application.

Feature extractor bolt: On receiving the input tuple from either the Query spout or the Feedback spout, the Feature Extractor applies the ensemble trees approach on extracting features. According to the description of the transformation algorithm in Sect. 3, for each input tuple, every decision tree generates a one dimensional sparse vector. Consequently, we can compress the output data of this bolt in an efficient way: to record the index of each non-zero element of the decision trees only, which helps to largely decrease the total data size to transmit through the network. In addition, each input tuple carries its sender information, say, the Query spout (respectively the Feedback spout), which is used by the Feature Extractor Bolt for choosing the correct downstream bolt, say the Classifier bolt (the Trainer bolt) to send the processed tuples.

Trainer bolt: Take the feedback data as the input, and based on the current coefficients of the LR model in the Classifier bolt, the Trainer bolt trains and updates the LR model. The training algorithm is very time consuming, thus to increase the throughput of the training phase, we apply the batch processing method. In particular, we allocate a separate memory space for buffering the input feedback tuples. Only when the total number of buffered tuples reaches a pre-defined threshold, will a dedicated thread be started to run the training algorithm. Finally, the Trainer sends these training results to the Classifier bolt after data serialization.

Classifier bolt: It takes input tuples from two upstream bolts. When the inputs are those user classification requests sent from the Feature extractor, it executes the classification job by using the LR model, and sends the results back to the user; when the inputs are from the Trainer bolt, it simply replaces the coefficients of the LR model with the input ones.

4.3 Interact with Spark

The supervised classification mostly consists of two phases, the training phase and the classifying phase. The former is delay insensitive, but shall be scalable and robust when the input data size is tremendous. The latter however, is delay sensitive, which means the response time for answering the user requests shall be as little as possible.

In our case, we have implemented the construction algorithm of the ensemble trees with Apache Spark. In the meantime, we utilize these ensemble trees to carry out fast feature extractions in the Feature extractor bolt. A technical challenge shows up when we try to achieve both: (a) keep the application topology running and unaffected; and (b) keep the ensemble trees up-to-date in real time.

HDFS (Hadoop distributed filesystem) [15] is therefore used to tackle this problem (as shown in Fig. 3). At first, we use Spark to run the ensemble trees construction algorithm, and serialize the newly generated ensemble trees onto the HDFS. During the initialization of Feature extractor bolt, the Storm metric API is called to register a temporal task, whose duty is to check the updates on the HDFS periodically. If there are updates, a separate thread will be started by the bolt to load the new generated ensemble trees from the HDFS in an asynchronous mode.



Fig. 3. Illustration of coordination among Storm, Spark and HDFS.

5 Experiment and Results

We implemented all proposed solution based on Apache Storm [12] v0.9.3 and Apache Spark [13] v1.2.0. The experiments were run on a cluster of 7 IBM servers, 4 of which were equipped with an Intel Xeon quad-core 2.53 GHz CPU and 32 GB of RAM, while others were equipped with an Intel Xeon dual-core 1.80 GHz CPU and 18 GB of RAM. We allocate one server to run as the master node, which hosts the Storm Nimbus. We choose 4 power servers to run spark, one as the master, the other three as slaves which have 8 GB of RAM and 16 cores. The Apache Kafka [14] is allocated on 3 servers and zookeeper 4 servers.

The dataset we use is CTR prediction from Kaggle Display Advertising Challenge. The first column is the label (click or not click) for each instance. There are 24,004,662 instances in the dataset which has 13 numerical features and 26 categorical features. In our experiment, we choose 70 percent of the dataset for training the hybrid model and the remaining 30 percent for testing purpose. We apply the hot encoding approach in the pre-processing phase to encode the categorical features, and set the minimum distinct value to be 4 million, i.e. only features with more than 4 million distinct values will be encoded and kept.

In the following, we show the experiments in which the hybrid model demonstrated above is used. We first run experiments to evaluate the accuracy of different classification schemes on the dataset. We then investigate on the training time of these schemes. At last, we running experiment on Storm to show the relationship between the query arrival rate and the average response time.

The classification accuracy We carry out experiments to demonstrate the effect of including tree features as part of the input to the linear model. In this experiment, we compare the five schemes:

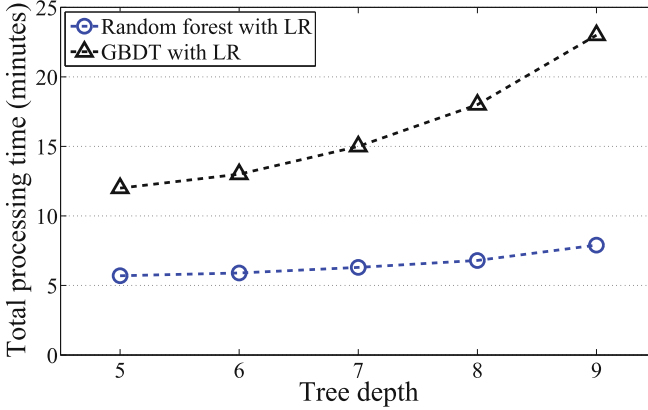
- (a) LR model with original features (LR only);
- (b) Random forest with original features (random forest only);
- (c) GBDT with original features (GBDT only);
- (d) LR model with features produced by GBDT (GBDT with LR);
- (e) LR model with features produced by random forest (random forest with LR).

The Area-Under-ROC (AUC) is used to test the performance of the models. AUC is a good metric when we are to measure the ranking quality without considering calibration. A larger AUC value means a more accurate model, hence the better classification results. AUC value of three different models on the testing dataset is listed in Table 1.

The total processing time for model training In this set of experiments, we investigate the total processing time for model training spent by different schemes. The flow of the training process is: at first, we use the numerical features and transformed categorical features to train the ensemble trees. Secondly, the ensemble trees are used to produce the binary vectors. Lastly, the binary vectors together with the numerical features and the encoded categorical features are used to train the LR model where the Limited-memory BFGS algorithm [16] was used.

Table 1. Classification accuracy comparison on five schemes.

Scheme	AUC (relative value)
Random forest only	94.5 %
Random forest with LR	98.9 %
LR only	96.2 %
GBDT only	95.3 %
GBDT with LR	100.0 %

**Fig. 4.** The total processing time changing with different tree depth.

The settings of Spark applications were as follows: the total number of executors are set to 40, driver memory is 2 GB, and executor memory is 8 GB. The total processing time here includes both the pre-processing time and the model training time. We study the sensitivity of the tree depth and the number of different trees to the total processing time of the two schemes, random forest with LR versus GBDT with LR. The results are plotted in Figs. 4 and 5.

According to the results in Table 1, Figs. 4 and 5, we observe that GBDT with LR earns the highest accuracy, however, it spends much longer training time than the random forest with LR scheme. The results indicate that the choice of GBDT with LR or the random forest with LR depends on the application context: when it is insensitive to the training time while sensible with accuracy, GBDT with LR is the better choice; Oppositely, random forest with LR will be more preferred.

Next, we test the performance of the online classification application run on the Storm with different arrival rate of the online requests raised by users. Figure 6 shows the curve of the corresponding average response time.

We observe from Fig. 6 that (a) the processing speed is really fast on Storm and the response time is less than 5 ms when the request arrival rate reaches 50 K per second; and (b) the average response time increases super-linear to the

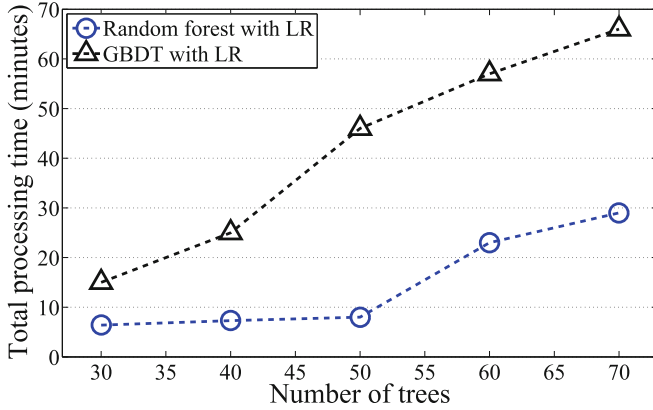


Fig. 5. The total processing time changing with the number of trees.

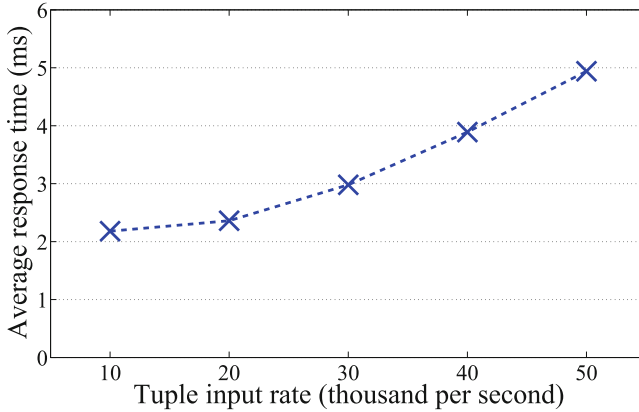


Fig. 6. The average response time changing with the tuple arrival rate.

arrival rate. This suggests that to keep the average response time under a certain threshold, far more resources shall be allocated at a higher request rate.

6 Conclusion

In this paper, we have studied and proposed a hybrid way (batching working with streaming processing) to implementing the online recommendation system with Apache Spark (for fast offline model training), Apache Storm (for quick answering users' online classification requests), and Hadoop file system (for data sharing and updating). This architecture helps enhancing the overall performance, in terms of the response time to the users' requests, the processing time for model training and updating, and the efficiency of the resource utilization.

Based on our hybrid framework, we also evaluated a group of online classification schemes. The experiment results imply that the choice on these different schemes shall depend on the using context, e.g. GBDT with LR earns the highest accuracy at the cost of long training time while random forest with LR earns the second highest in accuracy but spends less training time than GBDT with LR.

References

1. Huang, X.: Four directions of big data analytics in telecommunication industry. *J. Telecommun. Tech.* **6** (2013)
2. Neter, J., Kutner, M.H., Nachtsheim, C.J. Wasserman, W.: *Applied linear statistical models*. Irwin Chicago, vol. 4 (1996)
3. Richardson, M., Dominowska, E., Ragno, R.: Predicting clicks: estimating the click-through rate for new ads. In: *Proceedings of the 16th international conference on World Wide Web*, pp. 521–530. ACM (2007)
4. Graepel, T., Candela, J.Q., Borchert, T., Herbrich, R.: Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft’s bing search engine. In: *Proceedings of the 27th International Conference on Machine Learning (ICML- 2010)*, pp. 13–20 (2010)
5. Agarwal, D., Agrawal, R., Khanna, R., Kota, N.: Estimating rates of rare events with multiple hierarchies through scalable log-linear models. In: *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 213–222. ACM (2010)
6. Lee, K.C., Orten, B.B., Dasdan, A., Li, W.: Estimating conversion rate in display advertising from past performance data, uS Patent App. 13/584,545, August 2012
7. Menon, A.K., Chitrapura, K.P., Garg, S., Agarwal, D., Kota, N.: Response prediction using collaborative filtering with hierarchies and side-information. In: *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 141–149. ACM (2011)
8. Yan, L., Li, W.J., Xue, G.R., Han, D.: Coupled group lasso for web-scale ctr prediction in display advertising. In: *Proceedings of the 31st International Conference on Machine Learning (ICML-2014)*, pp. 802–810 (2014)
9. Stern, D.H., Herbrich, R., Graepel, T.: Matchbox: large scale online bayesian recommendations. In: *Proceedings of the 18th International Conference on World Wide Web*, p. 111120. ACM(2009)
10. He, X., Pan, J., Jin, O., Xu, T., Liu, B., Xu, T., Shi, Y., Atallah, A., Herbrich, R., Bowers S, et al., Practical lessons from predicting clicks on ads at facebook. In: *Proceedings of 20th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1–9. ACM (2014)
11. Gradient-Boosted Decision Trees, <https://spark.apache.org/docs/1.2.1/mllib-ensembles.html#gradient-boosted-trees-gbts>
12. Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J.M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham J. et al., Storm@ Twitter. In: *Proceedings of ACM SIGMOD*, pp. 147–156 (2014)
13. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M.J., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, pp. 2–2 (2012)

14. Kreps, J., Narkhede, N., Rao, J. et al.: Kafka: a distributed messaging system for log processing. In: Proceedings of 6th International Workshop on Networking Meets Databases (NetDB), Athens, Greece (2011)
15. HDFS, http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
16. Limited-memory BFGS, http://en.wikipedia.org/wiki/Limited-memory_BFGS

Web Technologies and Applications

APWeb 2015 Workshops, BSD, WDMA, and BDAT,
Guangzhou, China, September 18, 2015, Revised
Selected Papers

Cai, R.; Chen, K.; Hong, L.; Yang, X.; Zhang, R.; Zou, L.
(Eds.)

2015, XII, 171 p. 73 illus., Softcover

ISBN: 978-3-319-28120-9