Outbrain Click Prediction

Julien Hoachuck, Sudhanshu Singh, Lisa Yamada
{ juhoachu, ssingh9, lyamada } @stanford.edu

Abstract

In this paper, we explore various data manipulation and machine learning techniques to build an advertisement recommendation engine that prioritizes content to be presented to users. Companies like Outbrain have made it their mission to deliver quality content to their users and provide a platform for advertisers to reach their target audiences. Using Outbrain's click and user profile information, we manicured a data set using techniques such as binning and normalization. This data was used to train a logistic regression model and a random forest classifier to rank a set of ads on a given page in order of decreasing likelihood. We scored these classifiers using a mean average precision at 12 metric. In the end, we found that random forest performed the best and coupled really well with the binning technique used.

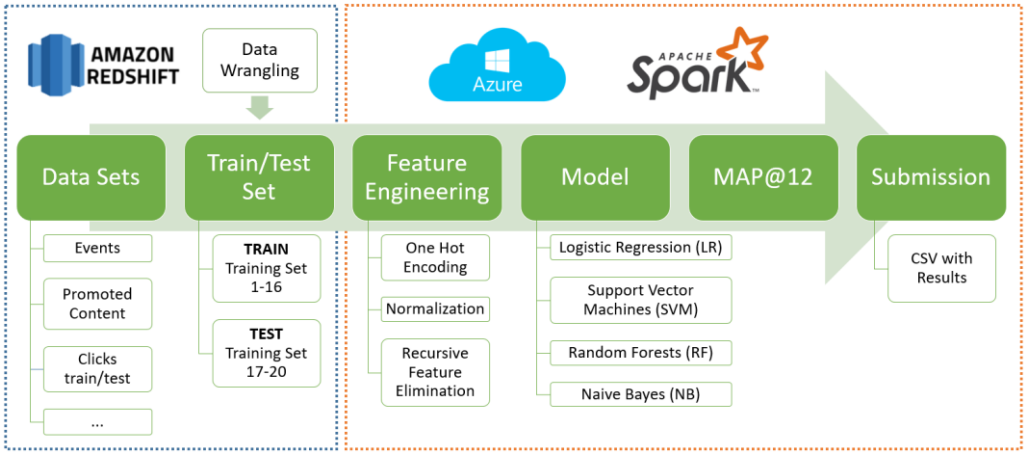


Figure 1. Machine learning pipeline for click prediction

Introduction

In modern society, the advent of technology has revolutionized the way people communicate and retrieve information, starting an era of constant information consumption. Mobile devices – laptops, tablets, and cell phones – are ubiquitous, providing a large scale of information to the public, such as technology, sports, weather, and international news. Due to the increasingly large amount of data that could be accessed, it is crucial to prioritize the content to present to users. Presenting optimal news that interest individual users, resulting in a higher likelihood of being clicked, improves user engagement and experience. The mission of Outbrain, a leading publishing and marketing platform, is to enrich the consumer with engaging content by building an advertisement recommendation engine. Machine learning algorithms could be used to predict users' behaviors and display pieces of content based their previous selections and features, ultimately providing a more personalized user experience.

To accomplish this task, Outbrain provides a large relational data (exceeding 100GB or 2 billion examples), providing a sample of users' page views and clicks observed across multiple publisher sites, platforms (desktop, mobile, tablet), and geographic locations between 06/14/2016 and 06/28/2016. The input to our algorithm is a set of key features that characterize the user, documents (originally viewed and promoted content), and the page view event (as shown in Table I). Most features were given numerical identifications, which is inappropriate for categorical features (i.e. platform: 1 = desktop, 2 = mobile, and 3 = tablet). These categorical features were one-hot encoded to properly treat them as categorical values rather than numerical values. Given a set of advertisements per document, we used logistic regression, support vector machines, and random forest algorithms to output an ordered list of advertisements in decreasing likelihood of being clicked for each document.

Machine Learning Pipeline

As illustrated in Fig.1., we used the Amazon Web Services (AWS) Redshift which uses Massive Parallel Processing to manage and query the large dataset, and Apache Spark on Microsoft Azure and Google Cloud platforms to train our models using distributed machine learning algorithms over the cloud. Initially, we were using a local computer and immediately realized the large computational power the task required. Given our time and resource constraint, it was required to setup this pipeline to process and iterate over this large dataset.

Dataset and Features

The Outbrain dataset provided a total of eight datasets:

- *Page views* describes features of all viewed pages, regardless of an advertisement being clicked.
- *Events* consists of features of pages viewed when one displayed advertisement was clicked.
- *Promoted content* provides advertisement features.
- *Clicks train/test* provides examples with labels to be used for training and examples without labels to be used for the Outbrain competition.

- *Documents meta* describes documents' metadata.
- *Documents entities*, *documents topic*, and *documents categories* provide mentioned entities (person, place, or location), topic, and taxonomy of categories of the documents, respectively.

According to [1], most data preprocessing take up to 80% to complete real-world data mining projects, especially those with high-cardinality categorical data fields such as this project. One of the main challenges was building the training and test sets for our models. Out of the 2 billion examples provided, we decided to exclude examples found in *Page Views* and only consider the 87 million examples contained in *Events*. These examples of page views that resulted in a click for one of the featured advertisements contain useful information to make click predictions. By using these examples, the first advertisement in the ordered list (output) represents the advertisement that we predict to be clicked for a particular document. In addition, *Documents entities* because it was too distinct, not providing much information. At one instance, training with entity as a feature prevented an algorithm from converging. Hence, features unique to *Page Views* (traffic source) and *Documents entities* (entity id and confidence level) were ignored. The remaining datasets could be mapped to each other using certain features as a key as illustrated in Fig.2.

TABLE I. Features provided by Outbrain (*n = 19*)

Users	uuid, **geographic location**
Documents	**Document id, ad id, source id, publisher id, publish time**, entity id & confidence level, **topic id & confidence level, category id & confidence level, advertiser id, campaign id**
Page View Event	display id, **timestamp**, **platform**, traffic source

Bolded = features used in our click prediction

Using AWS Redshift, we discovered that unique user id (uuid) were mostly distinct, indicating that observations were rarely made on the same user. Thus, it was impractical to use uuid as a feature. Also, display id was also not included in our feature because it is unique to each page view event. Display id represents a particular session of users viewing a document and allows us to group advertisements and features involved in the same event. However, it is not useful to include as a feature by itself. As a result, Fig.2 displays the dataset and features used for our prediction.

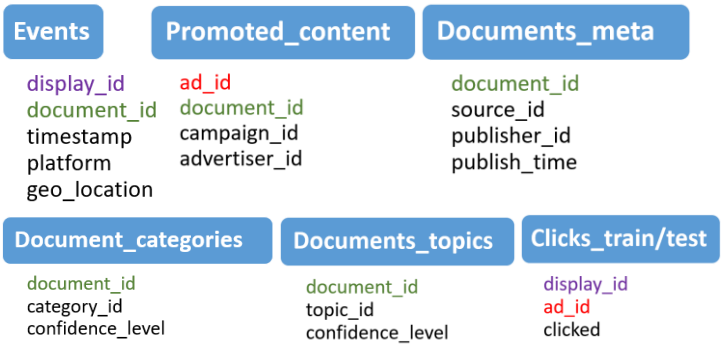


Figure 2. Datasets used for click prediction
Features used as keys are color-coded.

Feature Selection

As mentioned previously, most of our features are categorical with high cardinality. To overcome the issue of high cardinality, feature values of low frequency were grouped into a minority category. The threshold to determine if a feature value belongs in the minority category was strategically selected by observing its frequency percentile. The optimal threshold prevents information from being lost while maintaining a low cardinality of features. In the case of geographic location, Outbrain initially provided click data from 231 different countries. However, by determining the top 9 countries by popularity and "bucketing" other countries as a minority category, the size of the country feature was reduced from 231 to 10. In fact, the top 9 countries constitute 99% of the data, which verifies that information was barely lost while the cardinality of the feature was drastically reduced. This method was considered for 11 other features with high cardinality. Forming a minority group was necessary to run the models with limited computer memory. The threshold was selected with sound rationale; if all values demonstrated relatively equal significance, "bucketing" will result in loss of information and was avoided (i.e. event_topic_id, pc_category_id, and event_category_id). Table II presents the results of our "bucketing" method.

TABLE II. Results from "bucketing" method

Features	Original Cardinality	Cardinality after "Bucketing"
country	231	10
state	400	158
dma	212	148
advertiser_id	4174	300
pc_source_id	6990	200
pc_topic_id	300	198
pc_category_id	94	94
pc_publisher_id	882	120
event_source_id	4806	320
event_topic_id	300	300
event_category_id	90	90
event_publisher_id	484	300

pc = promoted content

To properly manipulate categorical features, one-hot encoding was used. For example, Outbrain provides platform information with numerical numbers (1 = desktop, 2 = mobile, and 3 = tablet). With one-hot encoding, this one feature was split into three features: is_desktop, is_mobile, and is_tablet. Each contained binary values (1 if the particular platform was used and 0 otherwise). Most of the categorical data was reformatted in this way. As a result, over 3,000 features (multiple GB of data) were ready for model fitting, emphasizing the need for setting up the pipeline.

Related Works

The task of ranking a set of ad ids given a particular session falls under the learning to rank problem. Learning to rank is common practice in providing recommendations for search engine results. There are two prominent approaches when it comes to learning to rank: pointwise and pairwise. In the pointwise approach, we train a probabilistic model that outputs a posterior probability for each ad id presented on the document viewed. The training data used to train this classifier is made up of all historical impressions. By definition, the model for the pointwise approach takes a single input at a time and minimizes the prediction error.

In the context of search engine document retrieval, this approach lacks any information regarding the relative ordering of the document and this is why a pairwise approach is taken instead. In [2], they show that the pairwise approach works much better than the pointwise approach. However, when trying to fit our problem into their framework, we found that the pairwise approach wasn't possible given our data set. A pairwise approach requires a ground truth that quantifies the relative importance of a group of ad ids given the document viewed which we could not infer from the data as well a set of features that lended itself well to creating a new feature from a pair of features [3]. The second restriction is usually not a problem when you have tf-idf vectors, pagerank, or other continuous features [4].

Unfortunately, we only had access to categorical features without much knowledge of what they actually meant. This being said, we had to use the pointwise approach. Since we had all categorical variables with high cardinality, we decided to explore algorithms that utilized ensembles of forests for the purposes of classification. This model was explored due to our not so successful attempt with linear discriminant model. In [5], the author points out that RF-based ranking yields good performance in general and they show how well it works in the ranking setting. The following sections covers some of these ideas as applied to our own learning problem.

Evaluation Metric: MAP@12

This competition required us to output a lists of ad_id(s) in order of decreasing likelihood of being clicked given a display_id (much like a session id). An appropriate error metric of such a list of recommendations with a sort of "relevance" score is MAP@K. In this competition, we were told to use MAP@12 which means mean average precision at 12. Unlike a conventional relevance ranking task, instead of a ground truth of relevance scores for each document given a query, we have a binary labeling of 0 and 1, where 1 is clicked and 0 is not clicked. The error metric is not as harsh as simply getting the answer right or wrong, but gives us some points if we are able to guess that the link would have been clicked second in the given context. The error metric is computed as follows:

$$MAP@12 = \frac{1}{|U|}\sum_{u=1}^{|U|}\sum_{k=1}^{min(12,n)} P(k)$$

Where *P(k)* = precision at cutoff k

$|U|$ = # of display_id

n = # of predicted ad_ids

After finding the average precision for each round of ad_id(s) per display_id, we compute the final score by taking the average over all the average precisions.

Methods

Logistic Regression (LR)

After applying the previously mentioned data transformations to produce a set of numerical features that can be used with a numerical discriminant model, logistic regression was our first choice of classifier. Since our data has binary-valued labels, $y \in \{0,1\}$, and we want to output the likelihood of an ad id being picked given a document view, logistic regression was the natural approach. The output of the classifier's sigmoid function can be interpreted as a probability and the likelihood that we want to maximize follows a binomial distribution. In our approach, we chose to use logistic regression with three different regularized cost functions, each with a different type of regularization: Ridge, LASSO, and Elastic Net. For both Ridge and LASSO, the regularization constant is λ, while Elastic Net has regularization constants of both λ and α. The following cost functions represent the previously mentioned regularized flavors of logistic regression:

$$argmax_\theta \sum_{i=1}^{m} \log p(y_i|x_i,\theta) - \lambda R(\theta)$$

$$L1: R(\theta) = \|\theta\|_1 = \sum_{i=1}^{m} |\theta_i|$$

$$L2: R(\theta) = \|\theta\|_2^2 = \sum_{i=1}^{m} |\theta_i|^2$$

Random Forest (RF)

Random Forest is a substantial modification of bagging that builds a large collection of de-correlated trees and then averages them [7]. Unlike logistic regression, it doesn't require a specific data type as input which works in our favor given the copious amounts of categorical features present in the data set. Another advantage to using this particular model, is the small number of hyperparameters required to be tuned. The algorithm is as follows:

B bag of trees and training data D of $\{(x_1,y_1),\ldots,(x_m,y_m)\}$.

1. for i=1:B

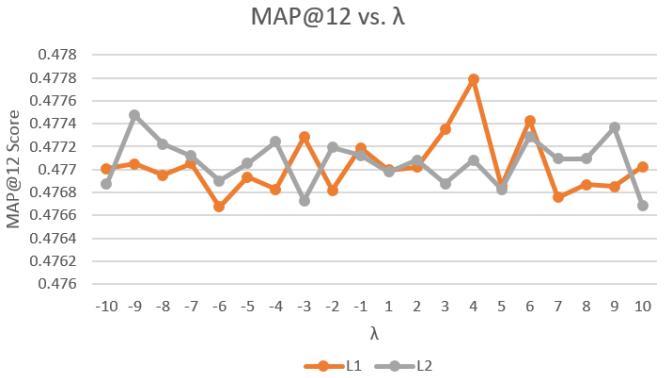
   Choose bootstrap sample D_i from D. Construct tree t_i using D; such that, at each node chose n random subset of features and only consider splitting on those features.

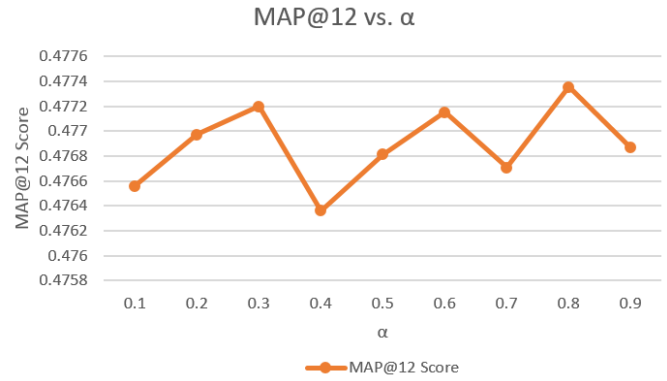
2. Once all trees are built, run test data aggregated predictor.
3. Given x, take majority vote (for y = 0,1) from different bags of tree.

Results

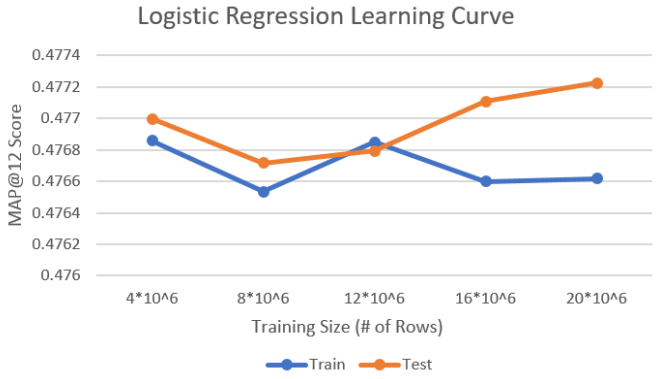
Logistic Regression (LR)

Each regularization technique gives us the opportunity to avoid overfitting as well as get more insight into which features will provide the best insight for our prediction task. Lasso regression provides a method of doing feature selection due to a sparse model being generated. Ridge regression has the ability to solve instabilities in vanilla logistic regression by solving the multicollinearity problem. And elastic net balances out the pitfalls of the two regularization techniques[paper?]. For our purposes, we found the optimal parameters for the model using stochastic gradient descent as opposed to L-BFGS since we had so much data and a time constraint. The plots below show the different values for MAP@12 via 5-fold cross validation as the regularization constant changes. In the case of elastic net, a regularization constant was held constant and alpha was adjusted to distribute the weight between the two regularization terms.



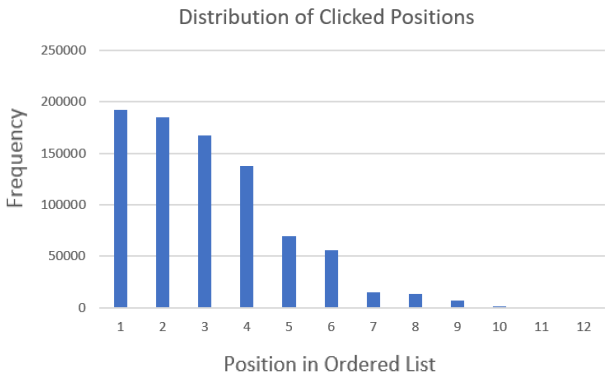


The model with the best score is LASSO logistic regression with $\lambda = 10^4$. Using this model, we iterated over different size training sets and produced the test and train scores to determine whether the model was overfitting due to the relatively low score it was producing during parameter tuning.



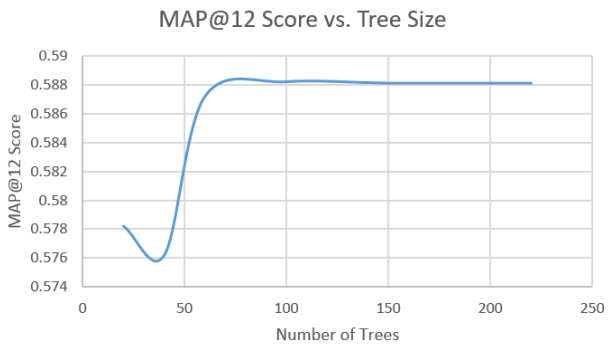
As the training set size increases, the test MAP@12 trends up while the training trend stays relatively constant. This is evidence of overfitting. On the other hand, it does suggest that substantial performance gain will not necessarily come from including more data during training or parameter tuning, but rather reworking features or choosing a model appropriate for the feature set we have. To further understand how our model is doing, the distribution of clicked positions over our predicted ordering was investigated using a custom

bucketing method.

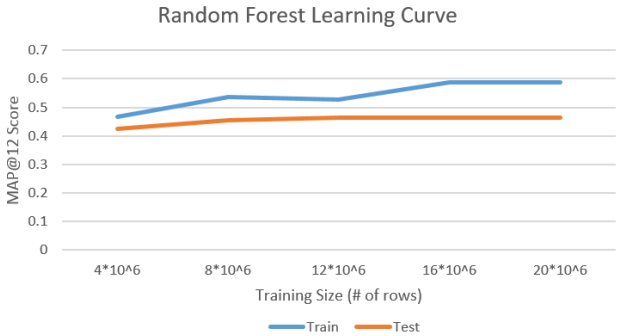


These results suggest that we are placing more truly clicked ads in the correct position rather than in any other location. Also, it should be noted that more than 50% of the truly clicked ads are placed in the top 3 positions.

Random Forest (RF)



We gradually increased the tree size, Random forests stabilize at 150 trees.



Since Random forest is quick to learn with little tuning, that explains why it the test score doesn't go up when we add more data.

Lessons Learned

First, we tried to run all the analysis and modeling on our personal computer, we start to get out of memory error, or it was taking excruciating long to iterate on smaller data set too. We learned, if a problem involves few GBs of data and features space >1500, we have to use distributed computing.

Discussion

Model Name	Score (Map@12)
Randomly	20%
Log Reg	47%
Random Forest	58%

Please note that the highest Map@12 score on Kaggle is 69% for this project.

Random forest is a clear winner as it works well with categorical features. Our score dramatically increased once we did one hot encoded features. We could potentially get a higher score if we had more time and computing resource and be generous with our feature bucketing strategy i.e. if we could include more of lesser frequent values as feature.

We didn't have a chance to analyze the temporal features; it would have been interesting to see what kind of documents users click during daytime or nighttime. We learned there exists a correlation between category of document users are presented with and the ads they clicked on. For example, if the original doc was Sports related, the user most likely clicked on Sports related promoted contents.

Conclusion

Given the large dataset and numerous amount of features, a lot of time was taken to curate a data set that would both be intuitive and not computationally expensive. The previous sections show that given our feature set and labels, the best ranker was random forests. Logistic regression was susceptible to the type of features we were presented with for ranking. Even after binning the features and tuning the models, we weren't able to achieve great results. On the other hand, binning the high cardinality features and training the random forest mode on our curated data set had a ~10% advantage over logistic regression.

In the future we would like to use factorization machines, cluster the high cardinality features using k-modes, and create continuous features that estimate the probability of different sets of features given a particular label. Factorization machines are known for doing well on large sparse data sets for recommendation. In fact, the top scores on Kaggle are a result of using this model. By using k-modes, we would be able to reduce the number of features as well as provide some context to the grouping of ad ids provided to the user. Finally, there exists some papers where using counts and estimated proportions of different feature combinations in the data set result in higher performance of classifiers like logistic regression.

REFERENCES

[1] D. Micci-Barreca, "A preprocessing scheme for high-cardinality categorical attributes in classification and prediction problems." *ACM SIGKDD Explorations Newsletter* 3.1 (2001): 27-32.

[2] Li, Cheng, et al. "Click-through prediction for advertising in twitter timeline." *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015.

[3] T. Y. Liu, "The Pointwise Approach," in Learning to rank for information retrieval. Berlin: Springer-Verlag Berlin and Heidelberg GmbH & Co. K, 2011.

[4] Cao, Yunbo, et al. "Adapting ranking SVM to document retrieval." *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2006.

[5] Ibrahim, Muhammad, and Mark Carman. "Comparing Pointwise and Listwise Objective Functions for Random-Forest-Based Learning-to-Rank." *ACM Transactions on Information Systems (TOIS)* 34.4 (2016): 20.

[6] Sharma, Neha, and Nirmal Gaud. "K-modes Clustering Algorithm for Categorical Data." *International Journal of Computer Applications* 127.17 (2015): 1-6.

[7] L. Breiman, "Random Forest," *Machine Learning*, vol. 45. Springer US, 2001, pp. 5–32.

[8] C. D. Manning, P. Raghavan, and H. Schütze, "Evaluation in Information Retrieval," *An introduction to information retrieval*. New York: Cambridge University Press, 2008.