Outbrain Click Prediction Competition Report

Quan Gan, Taha Raslan, Yuwei Tu

December 10, 2016

Abstract

This is a report for our participation in Outbrain Click Prediction Competition held on Kaggle. By the time of this writing, our model using Logistic Regression and some feature engineering ranked 30th on the Kaggle official leaderboard[1].

1 Motivation

With billions of Internet users, online media services have become commonplace. Prediction and recommendation for online media are fundamental problems in various applications. On the one hand, accurately predicting user behaviors can be helpful to improve user experiences through more intelligent user interfaces, on the other hand, user behavior prediction in online media is also strongly related to behavior targeting and online advertisement which is the major business for most consumer internet services.

Online advertising is a billions dollar industry that has served as one of the great success stories for machine learning. Understanding and predicting users click behaviors is a critical problem in online advertising. Sponsored search advertising, contextual advertising, display advertising, and real-time bidding auctions have all relied heavily on the ability of learned models to predict ad click-through rates accurately, quickly, and reliably.

Outbrain is the webs leading content discovery platform which pairs relevant content with curious readers in about 250 billion personalized recommendations every month across many thousands of sites. Our task is to predict which pieces of content its global base of users are likely to click on and improve its recommendation algorithm making more users uncover stories that satisfy their individual tastes. With good predictions, the conversion rate would be likely to increase, and both the advertisers and the advertisement platform (Outbrain in this case) can potentially see benefits.

[1] https://www.kaggle.com/c/outbrain-click-prediction/leaderboard. Our team name is GTR

2 Problem Definition

We formulate the problem as follows. First, we provide definitions of necessary entities according to description on Kaggle. Then we present the relationships and interactions between these entities. Finally we describe our objective.

2.1 Entity Definition

- *User*: An entity which actively browses pages. It has a unique identifier called UUID. Kaggle has anonymized the dataset so we cannot figure out any further detail regarding users themselves. We denote the set of users $\mathbb{U}$.

- *Document*: A document is a page, uniquely identified by an integer `document_id`. We denote the set of documents $\mathbb{D}$. A document has the following features:

  - *Source* and *Publisher*, represented by integer fields `source_id` and `publisher_id` respectively. Although they are treated as separate fields in the dataset, but documents with the same source also have the same publisher, so they can be treated as one feature theoretically. In our experiment they are treated as two separate features.
  - *Publish time*, represented by an integer timestamp `publish_time`.
  - *Category*, *Topic*, and *Entity*, each represented by an integral attribute `category_id`, `topic_id` and `entity_id`. Outbrain automatically detects these features for each document, and assigns a confidence level with attribute name `confidence_level` (a real number ranging from 0 to 1) for the most probable ones. We call these features *fuzzy categorical features* because the set of pairs of category/topic/entity and confidence levels forms a fuzzy set [7]. Documents are thus a kind of *fuzzy entity*.

- *Advertisement*: An advertisement is characterized by which document it resides in, which campaign it belongs to, and who its advertiser is, each represented by an integral attribute `campaign_id` and `advertiser_id`. It also has an integral unique identifier. We denote the set of advertisements $\mathbb{A}$.

- *Display*: A display is a scene where a certain user is browsing a document, or a relation between one user and one document, indicated by attributes `uuid` and `document_id`. It also contains additional information such as geographical location, platform (desktop, mobile or tablet), and timestamp. It also possibly includes the traffic source (via web search, internal

links, or from social network). We denote the set of displays $\mathbb{S}$. Note that multiple displays can share the same pair of user and document.

2.2 Entity Relationship Description

The dataset consists of several tables, describing the attributes of users, documents, and ads, and the relations among them. Figure 1 in Appendix shows an Entity-Relationship Diagram of our dataset. In short, we have the following interactions among users, documents, and ads:

- *Browsing cache*: Shown in Figure 1 as relationship "Browsed", the browsing cache contains pairs of users and documents, indicating which user browsed which document, as well as the browsing time, location (in the form of Country, State, and Designated Market Area (DMA)), platform, and traffic source.
- *Tri-relation between user, ad, and document*: The table of displays (relationship "In Display" in Figure 1) contains the record of every display with its user and document. Note that the user-document pairs in the set of displays are not guaranteed to appear in the browsing cache, and vice versa.
- *Clicks*: The table of clicks (relationship "Ad In Display" in Figure 1) stores whether an ad in a displayed is clicked or not. There are two such tables: one for training, and another for testing, which is the Kaggle official holdout set. The displays in the training set and test set does not overlap, which impacts how we decide to split the official training set for training and evaluation, as is discussed in Section 3.3.
- *The target document of an ad*: The relationship "Points To" in Figure 1 shows that each ad has a document it is linked to. We call that document the *target document* of an ad.

2.3 Task Definition

Now that we defined the entities and their relations, we can formulate the task into a data mining problem: Given a display, with related user and document, containing a list of ads, rank the ads by the likelihood of being clicked.

- *Dataset instance*: The dataset instance is a pair of display and ad. For each display s, we can figure out the user u and document d by inspecting s.`uuid` and s.`document_id`. Each user, document and ad has its own features, which will be discussed in Section 3.

- *Target variable*: Each dataset instance in the training set is labeled by a binary variable indicating whether the ad in the display was being clicked or not.
- *Task type*: Technically this is a ranking problem. We can also treat it as a classification problem, as most classification algorithms can also give a probability measure (e.g. Logistic Regression), or some certainty measure which is correlated to probability (e.g. SVM)
- *Temporal information*: It is worth noting that the displays and browsing records have timestamps. Strictly speaking, we need to treat the dataset as a time series so as to properly reflect the trend of data generation process, such as user preferences. In the competition, Outbrain splits the dataset into training set and official holdout set *uniformly at random*, based on an assumption that the data generation process does not vary over time. So we chose not to consider the dataset as a time series. In our work we did not even use timestamps as a feature, while still being able to get reasonably good results.

3 Data Preparation

In this section, we will discuss how we process each entity and attribute into vectors and/or matrices so that we can apply existing models to make useful predictions. We will discuss general strategies first, followed by specific methods on each attribute of each entity and/or relationship, as well as how we partition the training set for offline evaluation before submission to Kaggle.

3.1 General Strategy

- For each categorical variable C, we dummy it out into $|C|$ indicator variables and treat these variables as a single one-hot vector, where $|C|$ is the number of possible values for C.
- Encoding fuzzy attributes into vectors is similar to encoding a non-fuzzy attribute. For a fuzzy attribute, we construct a zero vector with the number of dimensions the same as number of possible values for that attribute, then for all values with non-zero confidence level, we set the corresponding elements to those confidence level values.

3.2 Specific Feature Encoding

- *Documents*: For each document, we encode its source and publisher into two one-hot vectors. We encode document category, entity and topic into three vectors by the method for encoding fuzzy attributes. These vectors are then concatenated together into a single *document feature vector* (DFV).

- *Ads*: Each ad has a target document, so we can find the DFV of that target document, and view it as a feature for an ad as well, which we call the *ad document feature vector* (ADFV). We also encode the campaign and advertiser of an ad into two one-hot vectors. We then concatenate the two one-hot vectors with the ADFV to form an *ad feature vector* (AFV).

- *Users*: Since the dataset is anonymized, the only thing we can exploit for the users is their browsing records. We have several options to extract features for users:

  1. The first is to treat every user as identical, and encode their UUID into a one-hot vector. This method has several disadvantages: not only the dimensionality of this vector is large (equal to the number of users $|\mathbb{U}|$, over 20,000,000), it also cannot capture the similarity between users.
  2. Another option to create a $|\mathbb{D}|$ dimensional vector and put the number of browses on each document for this user into the corresponding cells. We call this kind of vector *user browse vector* (UBV). By doing this we reduced the dimension into $|entitysetD|$, which is around 3,000,000. It can also capture some similarities between users.
  3. By treating the number of browses as implicit "ratings" a user gives to a document, we can perform *Matrix Factorization* on the user-document matrix, whose entries are the number of browses, and treat the factor vector for a user as a feature. We call that factor vector the *User Factor Vector* (UFV). The same can also apply for documents, but we believe that the factor vectors are not as good predictors as the documents' own attributes, so we did not include the factor vectors for documents.
  4. We can also make the average of the DFVs of all documents the user has browsed as the *user profile vector* (UPV), which may be a good representation of user preference.

  In our experiments of Linear Models, we tried all options to find the best feature for the users.

- *Displays*: Currently we ignore the attributes of the displays themselves. This means that we disregard the information of timestamp, geolocation, platform and traffic source.

3.3 Dataset partitioning

We partition the official training set into an *offline training set*, an *offline validation set*, and an *offline test set*. For linear models, the validation set is for tuning the hyperparameters.

Recalling from Section 2.2 that the displays in the official training set and test set do not overlap, we had to group the training set by displays, then uniformly split the groups into offline training set, validation set, and test set with a ratio of 8:1:1.

4 Models and Evaluations

4.1 Evaluation Metric

Kaggle uses the *Mean Average Precision at Cutoff 12* [8] metric (MAP@12) for evaluation[2]. The formulation of *average precision at cutoff 12* on display s is:

$$AP@12_d = \frac{1}{\sum_{i=1}^{\min(12,n_s)} r_i} \sum_{i=1}^{\min(12,n_s)} r_i \left(\frac{\sum_{j=1}^{i} r_j}{i} \right) \tag{1}$$

where n_s is the number of ads in display s, and r_i is 1 if the user clicked on the i-th recommended ad, or 0 otherwise [5]. MAP@12 is the mean of AP@12 across all displays.

This metric is particularly useful in evaluating recommender systems in cases where only a limited number of recommendations can be made, because it does not overly penalize the model for ranking correct predictions slightly lower, comparing to other binary classification metrics such as precision and recall. We follow this metric in our model selection process.

4.2 Baseline: Rank-by-Popularity

It is natural to recommend the most popular item to users, regardless of their interests. From this intuition we have the baseline model called **Rank-by-Popularity**: for each ad, compute the empirical probability, or the ratio between number of clicks and number of show-ups. More formally, for an ad with ID a:

[2]https://www.kaggle.com/c/outbrain-click-prediction/details/evaluation

$$\text{score}\,(A) := \frac{(\#\text{ records in training set with Ad ID } a \text{ and Clicked labeled 1}) + \alpha}{(\#\text{ records in training set with Ad ID } a) + \beta}$$

where α and β are smoothing hyperparameters.

The formula becomes add-one smoothing when $a = 1$ and $b = 2$. However, doing so implies that the "prior" probability we give to all ads not appearing in the training set is 0.5, which is not very appropriate because people in real life do not click on ads that often. Setting $a = 0$ gives a more conservative estimate, where the "prior" probability is 0. The value of b is not important since we only care about the order of ad scores. In our experiments we evaluated both hyperparameter settings on validation set. Since **Rank-by-Popularity** does not involve any training, we can directly evaluate the performance on validation set to pick the best hyperparameters.

Rank-by-Popularity performed surprisingly well in terms of MAP@12: it achieved 0.645 on our offline test set, with hyperparameter $\alpha = 0$ and $\beta = 2$, although the performance did not differ more than 0.01 when varying a and b. As a side note, we did not make submissions based on this model, but it is said to achieve 0.635 on official test set[3], much higher than random guess method, whose MAP@12 is 0.486[4].

4.3 Linear Models

The next model family we tried was Logistic Regression. We trained and evaluated logistic regression models with L2 regularization and different features. The score of an ad in a display is simply given by the dot product between the feature vector and the weights of the model, plus the intercept of the model.

In order to evaluate the AP@12 for each display in the validation set or test set, the model predicts the rankings of the ads in that display by ranking them according to their score, and computes the AP@12 metric according to Equation 1. The MAP@12 on the validation set or test set is just an average of AP@12 over all displays in that set.

The regularization coefficient is determined by grid search among {0.00001, 0.0001, 0.001, 0.01, 0.1} and picking the one with the best MAP@12 on validation set. We report the MAP@12 on offline test set afterwards. The experiment setup is described in detail in Appendix.

Model name	MAP@12
Rank-by-Popularity	0.645
LR-Naive, $\lambda = 0.001$	0.646
LR-CTR, $\lambda = 0.0001$	0.648
LR-UBV, $\lambda = 0.001$	0.646
LR-UBV-UPV, $\lambda = 0.001$	0.648
LR-UBV-Factors, $\lambda = 0.0001$	0.647
LR-All, $\lambda = 0.001$	0.650
LR-Interactions-All, $\lambda = 0.001$	0.672

Table 1: MAP@12 for logistic regression models with different features on our offline test set. λ is the L2 regularization coefficient determined by grid search. The meaning of each model name is described in Section 4.2 and 4.3.

4.3.1 Naive Features

Table 1 shows the performance of Logistic Regression with different feature engineering regarding to a single entity, in comparison with the baseline **Rank-by-Popularity** method. Each model name prefixed with **LR** represents a logistic regression model with the following features for each training instance:

- **LR-Naive**: Treat users as identical (encoded into one-hot vectors with $|\mathbb{U}|$ indicator variables), and include the DFV of the document in the display and the ADFV of the ad.
- **LR-CTR**: As above, but with an additional predictor which equals to the empirical probability computed in the **Rank-by-Popularity** model, as we believe that it is easier for the model to exploit the historical popularity of an ad by explicitly giving the ratio.
- **LR-UBV**: Replace the one-hot vector for each user in **LR-Naive** by the UBV of each users. We did not normalize the vector with number of browses as linear models usually can handle different scales automatically.
- **LR-UPV**: Adds UPVs as another predictor to **LR-UBV**.
- **LR-Factors**: Adds UFVs obtained by Matrix Factorization described in Section 3.2 to **LR-UBV** as another predictor.
- **LR-All**: Includes everything, i.e. the DFV of the document in the display, the AFV of the ad, the historical CTR of the ad, as well as the UBV, UPV and UFV for the user.

We can see that even with all of these features, the logistic regression only made marginal improvement over the **Rank-by-Popularity** model.

[3]https://www.kaggle.com/clustifier/outbrain-click-prediction/btb-0-63523-evaluation
[4]https://www.kaggle.com/c/outbrain-click-prediction/leaderboard

4.4 Representing Interactions

Feature interactions can also be good predictors. For instance, the similarity between UPV and ADFV may be significant in predicting ad click-throughs, as a higher similarity means better chance for that user to be interested in the content of that ad. Among similarity measures, *cosine similarity* is the most common measure for determining the similarity of two documents in terms of their subjects [4]. We also believe that in case of ad click predictions, the similarity of some subjects would contribute more to the likelihood of click-through than others.

From this perspective, we introduce the element-wise product of UPV and DFV, DFV and ADFV, and UPV and ADFV as new predictors, as a generalization of cosine similarity, since cosine similarity is essentially an average over element-wise products between two normalized vectors. Note that both UPVs and ADFVs have the same number of dimensions as DFVs, since a UPV is an average over DFVs, and an ADFV is essentially the DFV of the target document of an ad.

LR-Interactions-All is a logistic regression model which has all the features in **LR-All**, as well as the three new predictors. It achieved 0.672 in our offline test set, a comparatively big improvement over the **Rank-by-Popularity** model. It also achieved 0.664 on Kaggle official holdout set, which is ranked 30th, or in the top 10% among the participants who beat the **Rank-by-Popularity** model, by the time of this writing[5]. Interestingly, this model also beats some participants using more sophisticated model such as Factorization Machines [3], which could capture the feature interactions by its nature[6].

5 Deployment of our Model

Our **LR-Interactions-All** model relies on the browsing caches, and therefore the records of user browses to compute UBVs and UPVs, and relies on the detection provided by Outbrain to construct the DFVs and ADFVs. Also we need a mechanism of recording user clicks on ads to get the ground truths for training our model.

The dependence on user profiles also means that the model is not able to make useful recommendations towards new users with little browsing history recorded, so we need to leave some time to observe the user behavior, aggregate

[5]https://www.kaggle.com/c/outbrain-click-prediction/leaderboard. Our team name is GTR

[6]https://www.kaggle.com/qqgeogor/outbrain-click-prediction/keras-based-fm. The author himself achieved 0.65445, but it is worth noting that this post inspired some participants to get some of the top rankings, including the number one solution which reportedly used the Field-aware Factorization Machines [2].

his or her browsing history, evaluate the UPV and/or UBV before actually making personalized recommendations.

Often time, we have to combine the **LR-Interactions-All** model with **Rank-by-Popularity** model, as **Rank-by-Popularity** is a good fallback method for new users. Doing so would introduce a *positive feedback loop*, though, since more popular ads are showing up more often, hence getting more chance of being clicked in general, so that the chance of getting ground truths for not-so-popular ads becomes even less.

The entire advertisement system is a dynamic system: user behaviors may change, new users will come in, new documents will pop up, and new ads will be contracted. Therefore, we need to re-evaluate the features including UBVs, UPVs, DFVs and AFVs, and re-train the model occasionally, possibly once per some fixed period.

Once trained, the model is able to rank all the ads and select the topmost ads with the highest scores for display in limited space. Evaluating over all the ads would probably be time-consuming, even though the inputs for our model are usually sparse vectors, so a better application is to sample some ads in the database, and pick the ones with the highest scores to display.

6 Discussion

Our model still has a large room for improvement. For instance, we did not consider informations such as timestamps, geolocations, traffic sources and browsing platforms as features. While traffic sources and browsing platforms are easy to add as they are categorical variables, we would have to scale down the timestamp values if we want to try more sophisticated model families, although we can leave it as-is in linear models. Geolocations can be converted to longitudes and latitudes using tabulated data or third-party APIs.

Clustering the users, documents and ads into groups may also improve the model by possibly cleaning up noises and reducing the number of dimensions.

It may also be worthwhile to try other model families, including Field-aware Factorization Machines [2], which are already reportedly good in click-through rate prediction. Neural networks is also an option. Since both our dataset and model are large (because of the number of dimensions of the input vector), we may need to use Parallelized SGD [9] or Downpour SGD [1] to effectively train these models. Unfortunately due to engineering difficulty we are not able to run more sophisticated models for this write-up in time.

Appendix

Entity-Relationship Diagram

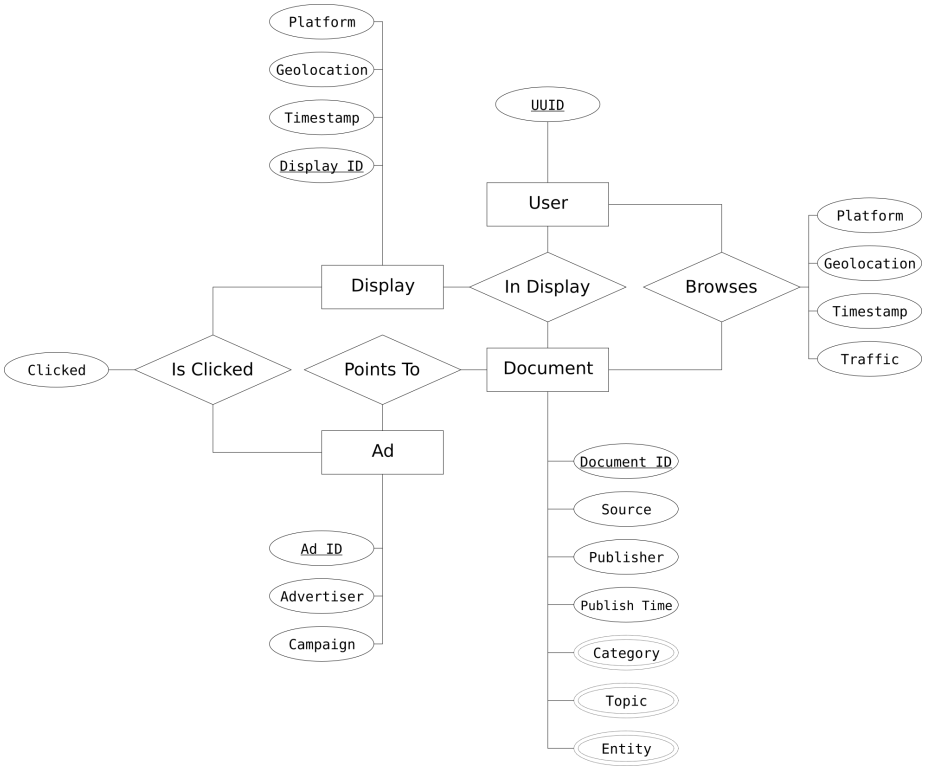

Figure 1: The database schema in Outbrain Click Prediction, in terms of Entity-Relation Diagram. The double dashed ovals are the so-called *conjunctive fuzzy attributes* [6], meaning that the attribute can take multiple values with some degrees of membership (or confidence).

Experiment Setup

The dataset consists of more than 80,000,000 training instances, as well as a browsing cache with more than 2 billion records. To deal with such large data, we leveraged a 48-node Spark cluster each with 64GB of RAM to preprocess the data, including user profile computation and Matrix Factorization, as well as training the logistic regression models.

We also tried evaluating Linear SVMs on the Spark cluster. Due to time and software limitations we are unable to get Kernelized SVMs working, and we found that SVMs somehow performed worse than random guess. This is left to be investigated. Because we have a large number of dimensions, we cannot

get random forests to work on Spark cluster.

Acknowledgments

In our work Quan was responsible for data preprocessing, feature engineering and exploring how to use PySpark in general. Yuwei prompted us to participate in this competition, and provided insights on business understanding and possible deployment issues. Taha explored the possible models and learning algorithms to use, as well as how to train and evaluate the models.

References

[1] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, MarcAurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *NIPS*, 2012.

[2] Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pages 43–50, New York, NY, USA, 2016. ACM.

[3] Steffen Rendle. Factorization machines with libfm. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 3(3):57, 2012.

[4] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[5] Andrew Turpin and Falk Scholer. User performance versus precision measures for simple search tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 11–18, New York, NY, USA, 2006. ACM.

[6] Li Yan and Z.M. Ma. Formal translation from fuzzy {EER} model to fuzzy {XML} model. *Expert Systems with Applications*, 41(8):3615 – 3627, 2014.

[7] L.A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338 – 353, 1965.

[8] Mu Zhu. Recall, precision and average precision. 2004.

[9] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. Parallelized stochastic gradient descent. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural*

Information Processing Systems 23, pages 2595–2603. Curran Associates, Inc., 2010.