Personalized Web Search using different ranking algorithms

Biman Gujral
bgujral1@jhu.edu

Rujuta Deshpande
rdeshpa3@jhu.edu

Abstract

Our project aims at personalizing web search rankings of a user by analyzing user search history from Yandex's search engine logs. We have built a total of 73 features on the log data, different combinations of which are run on Learning to Rank algorithms. We analyze the importance of different features through the NDCG score obtained for different combinations. The algorithms used are - Random Forests, LambdaMART, RankNet and AdaRank for a each set of features.

1 Introduction

Learning to Rank is a technique that constructs ranking models for Information Retrieval Systems. The training data consists of a set of query-document[1] pairs. Each query-document pair forms one data point and associated with a feature vector. Each of the documents in query results is also assigned a relevance score. They are ranked in decreasing order of these relevance scores. The task of a Learning to Rank model, is to assign a relevance score to each query-document pair in the test data and thereby, create a ranked list of the documents returned for a particular query. Personalized rankings re-rank the general results returned by a search engine in an order that would be suited to a given user's requirements. This preference is learnt from the user's past search history, by means of user-specific and generic features.

[1]Document and URL is used interchangeably

1.1 Background

Our project is derived from a Kaggle competition sponsored by the Russian search engine Yandex. We borrow heavily in terms of approach from the winning team of the competition (Masurel, 2013). However, unlike the winning team, we aim to do a feature-algorithm analysis. We have sampled the 16 GB training data in order to report personalization results for 16 user profiles. But, a larger history data is created since it also provides feature values for the general search patters apart from user-specific history.

2 Related Work

Personalized web search has been approached in various ways. Some approaches, aim at using only the user's long-term history. (**?**). User clicks have been observed to be most informative in predicting rankings (**?**). The satisfied-clicks(SAT-clicks) have become a standard in evaluating personalized search systems. Some literature (**?**) has shown that not all queries can be personalized. Navigational queries, which tend to have the same URL clicked by a majority of users do not benefit from personalization.
Our approach uses only long-term history for each user. It does not include the current user session for ranking. According to our proposal, we intended to take a weighted combination of anterior history and current session history for ranking. But, we had to limit to just the anterior history due to the large volumes of data which were sampled in a slightly different way from the original data.

2.1 Algorithms

We use Learning to Rank algorithms for this web search task. They fall into three broad categories (**?**)Wikipedia: Learning to Rank):

2.1.1 Pointwise Algorithms

In pointwise algorithms, each query-document pair has a numerical score associated with it. The algorithm transforms this into a regression problem of predicting a score given a query-document pair. If the scores take values from a finite set, this can even be a classification problem. Example: Random Forests

Random Forests learn a large number of decision trees based on random sample sets of the data(with replacement) and for any test sample, they output the mode value returned by running that sample through all the trees. In a random forest tree, unlike regular decision trees, the node is split based upon a random subset of features from the entire feature set.

2.1.2 Pairwise Algorithms

In the pairwise approach, each pair of documents returned for a query is selected. Between the two documents in a pair, the algorithm determines the more relevant document. The goal is to minimize the inversions in a ranking. Example: RankNet

2.1.3 Listwise Algorithms

Listwise algorithms either find the optimum score or minimize inversions. However, they average over all queries in the training data. Example: AdaRank, LambdaMART

2.2 Evaluation Metric

We are using NDCG (Normalized Discounted Cumulative Gain) as our evaluation metric since it is suitable to measure performance of a search engine. It is a metric between 0 and 1 that evaluates the ranking order. It is given by:

$$NDCG_k = \frac{DCG_k}{IDCG_k}$$

where k denotes documents uptil rank k and DCG is Discounted Cumulative Gain, given by:

$$DCG_k = \sum_{i=1}^{k} \frac{2^{rel_i} - 1}{log_2(i+1)}$$

where rel is the actual relevance of the document provided by labels and i is the rank given by the algorithm. Thus, a highly relevant document ranked later in the list results in penalization and a low DCG. The Ideal DCG or IDCG gives the best ranking in accordance with the relevance values. Therefore, NDCG = 1 when the ideal ranking is obtained.

3 Methodology

In order to personalize user rankings, we build a set of 73 features. We then, turn on/off different features and run the ranking algorithms on various combinations of these features. We compare the results returned by different ranking algorithms, which is discussed in detail in the result section. The features generated are similar to the Kaggle winning team's solution. However, as discussed above, our aim is differs in that we do a comparative analysis on sampled data. Our implementation is explained in detail in the following section.

3.1 Data Preparation

Kaggle provided about 16 GB of training data that covered 27 days of search log history. The format of the data is as follows:

Session metadata (TypeOfRecord = M):

SessionID TypeOfRecord Day USERID

Query action (TypeOfRecord = Q or T):

SessionID TimePassed TypeOfRecord SERPID QueryID ListOfTerms ListOfURLsAndDomains

Click action (TypeOfRecord = C):

SessionID TimePassed TypeOfRecord SERPID URLID

In the above format, the SERPID is the search result page's ID. Queries are uniquely identified by QueryID and its terms are given by the ListOfTerms. The ListOfURLsAndDomains has a list of 10 comma-separated URL-Domain pair returned for a query. Therefore, we will have 10 feature vectors for each query.

An example data format has been shown below:

8 M 6 5

18 0 Q 0 3771444 1823379, 1235163, 1061370, 1991523, 2140938 7885541, 943850 17062586, 1787472 17966961, 1887339 37836161, 3478295 52887117, 4315816 12350524, 1236243 39236857,

3559506 36287475, 3400946 41939085, 3715627 53975922, 4374403
18 19 C 0 12350524

3.1.1 Data Sampling

We sampled the data as our system is limited to 8 GB of RAM with dual cores. We split their train data into history, train, development and test sets for a specific subset of 28 users. This led to a significant reduction in the file size with history becoming about 120K, train 60K, development and test each with 30K file size.

Since we are doing a user-specific personalization, we wanted enough common user occurrences across history, train, test and dev. But, the data tends to have a single user's sessios together in the train file which prevented us from serially splitting the file. Hence, we randomly assigned a user's session to one of history/train/dev/test/. This has led to one limitation - we cannot make a clear distinction between immediate history and anterior history. The component of 'past behavior' which could be broken down into recent past and history gets reduced to a user's generic search behavior from history logs.

3.2 Feature Engineering

We aim to engineer features and estimate their importance through NDCG score obtained on different combinations. We generate a feature vector for each query-document pair in the input data. Here, document refers to a URL,domain combination. For each vector, we calculate a relevance label. The relevance label is based upon two factors - whether a user clicks on the URL and if he clicks, what is the dwell time for that URL (time spent on that URL- i.e. time between that click and the next click). The relevance labels are calculated as in table below:

1	URL missed
2	URL skipped
3	URL clicked with satisfaction 0
4	URL clicked with satisfaction 1
5	URL clicked with satisfaction 2

For any query, a list of URLs is returned. The log file contains a field called time passed, which indicates how much time has passed from the beginning of that session, up till that query/click. If a URL is clicked, we can find the dwell time for that URL as the difference between that click and the next click. The last click of the session is assumed to provide satisfaction and given a relevance label of 5.

A dwell time between 50 and 300 units of time, indicates a relevance score of 4(click with satisfaction 1), dwell time of more than 300 units(satisfaction 2), indicates relevance score of 5 and a dwell time lesser than 50 units(satisfaction 1), indicates relevance score of 3. We assign the missed and skipped scores, based on the *Cascade Hypothesis* (**?**)ref 9). This states that, all URLs above a clicked URL have been examined (and skipped) and all URLs below the lowest clicked URL haven't been examined (and missed). Thus, all skipped URLs are given scores of 2 and all missed URLs are given the lowest score or score 1.

This way, we assign a relevance label to each query-document pair. We have broadly divided our features into three categories:

Query features - These features are not specific to any user in particular. Features falling in this category are:

Initial rank of a document for a particular query. This rank is the one originally returned by the non-personalized search engine algorithm. It has all the information we have about page-rank or document similarity that Yandex might have used to compute the original rank.

The frequency of a particular query. This gives us information as to how popular a particular query is

The number of terms in a query.

The position of the query in a session.

Aggregate features - These features can be user specific or generic, however, they are computed over all the history data, based on a certain predicate. A part of these features is basically a probability of the document being clicked, missed or skipped, conditioned on a predicate while rest are the Mean Reciprocal Rank of these misssed/skipped/clicked URLs . When a particular query-document is encountered in the training or development set, we consider a set of predicates to filter logs and produce features on this subset of logs. The predicate is a conjunction of conditions on :

- the url (same url or same domain)
- the user submitting the query (same user or any

user)
- the query (same query or any query)
Thus, if we encounter a query-document combination $< q_1, d_1 >$ for some user u we construct features by conditioning on each of the 8 predicates obtained from the above conditions as *PR User u, queried for* q_1*, got a document* d_1. So, the feature we will calculate is:

$$P(outcome = \ell | PR) = \frac{Count(\ell, PR) + p_\ell}{Count(PR) + \sum_{\ell'} p_{\ell'}}$$

There are total of 5 possible outcomes - the user either missed or skipped the URL returned, or he clicked it with a satisfaction of 0 or 1 or 2. Given a query-document, we get a probability of the number of times the document was missed (or clicked or skipped) by dividing by the total number of times the document appeared in the subset of filtered logs as per the predicate. For example, given the example predicate above, an outcome of 'skipped' would be probability that this URL was skipped given that the same user queried for the same query previously. This helps measure results that are not user-specific by calculating the probability that any user will skip that URL or URLs in the same domain. This probability was smoothed using additive smoothing because the subset returned on this predicate might be empty if that URL is never clicked. The additive smoothing assumes that all URLs have been missed once. Hence, p_ℓ represents a prior as is defined as $p_\ell = 1$ if ℓ is Miss and 0 otherwise.
Moreover, the conditional probabilities do not take into account the original rank of displays. This rank, adds a large bias, as a document ranked 1, is more likely to be clicked than one ranked 10. Hence, another feature called MRR has been calculated for the Miss, Skip and Click occurrences. It is defined as the reciprocal value of the harmonic mean on the URL of the ranking (?)Masurel:13).
The MRR is calculated as:

$$MRR(\ell, PR) = \frac{\sum_{r \in R_{\ell,PR}} \frac{1}{r} + 0.283}{Count(\ell, PR) + 1}$$

Thus, we get the harmonic mean of the URL ranks. Additive smoothing is done follwing the assumption that every URL was displayed at lease once. 0.283 signifies the inverse of the prior on the rank of this URL. **User specific features** - These features try to learn a user's click habits. They are:
Number of queries issued by the user.
Average number of terms in that user's queries.
Number of times the user clicked on returned URLs that were in ranks $1, 2$ or in ranks $3, 4, 5$ or in ranks $6, 7, 8, 9, 10$. These lead to 3 features per user.
We generated all of these features per user and created a user's feature vector which was added as the first line of each use specific file. The format of our file is:

numQuery:11 **numAvgTerms**:2.36363636364 **numClicks6**:0 **numClicks35**:6 **numClicks12**:9 ('2452324', '14603791', '2868731', '55955179,4457118') **rank**:1 **pos**:1 **terms**:5 **frequency**:0 **score**:2
aggr:[[1.0, 0.0, 0.0, 0.0, 0.0, 0.283, 0.283, 0.283], [1.0, 0.0, 0.0, 0.0, 0.0, 0.283, 0.283, 0.283], [1.0, 0.0, 0.0, 0.0, 0.0, 0.283, 0.283, 0.283], [1.0, 0.0, 0.0, 0.0, 0.0, 0.283, 0.283, 0.283], [1.0, 0.0, 0.0, 0.0, 0.0, 0.283, 0.283, 0.283], [1.0, 0.0, 0.0, 0.0, 0.0, 0.283, 0.283, 0.283], [1.0, 0.0, 0.0, 0.0, 0.0, 0.283, 0.283, 0.283]]

The first row specifies the user feature. Follwoing that is a series of query-document pair specific features per row. The first tuple in every row specifies (UserID, SessionID, QueryID, <URL,Doman>). Following that, are a set of key-value pairs for query and aggregate.
However, we want to be able to turn features on and off, as well as transform this data into the format of LETOR data set to use it with RankLib library.
For this we define a configuration file that has all the feature keys defined, with a value of 1 or 0 against it. This indicates that feature the feature is considered(1) or not(0). Since the aggregate features are many, we defined a comma separated string against it. A string of 0,0,1,0,0,0,1,0 indicates that we consider the predicate with value 3 or binary 011 that translates into (Same User-Same Query-Same domain) and so on. Under that we define another 8 bit string that specifies which of click/skipped/missed probabilities and MRR to calculate as features for each of the predicates. When both strings are all 1s, we get a total of 64

aggregate features.
We then run a script that reads both the configuration file and the user defined feature file to create a LETOR data set input file per user for the specific combination of features. Now, we can actually run the algorithms on our data.

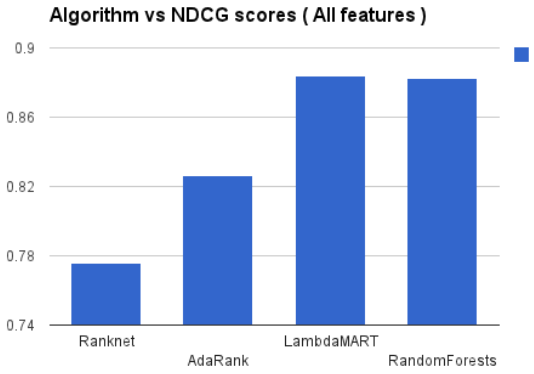
3.3 Details of Tests

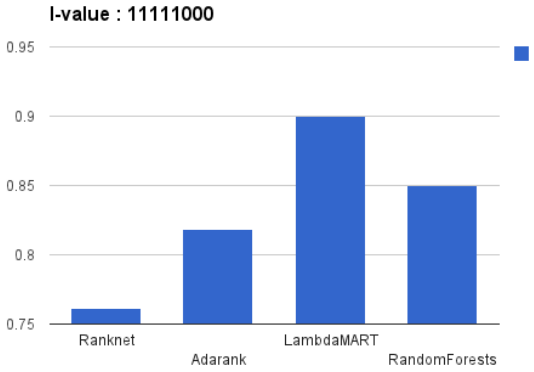
We ran the four algorithms under 6 sets of feature combinations. The feature combinations are:
-All features turned on (Total 73 features)
-Query Features turned off (position of query in session, number of terms in query, frequency of query) (Total 70 features)
-Aggregate Features turned off (Total 9 features)
-User specific aggregate features on (Vector: 00001111)
-Probability/MRR of Skip, Click, Miss (Vector: 00000111)
-Probability/MRR of Skip, Click, Miss (Vector: 11111000)

3.4 Results

Thus, from the results, it can be seen that LambdaMART has performed better than all other algorithms, followed closely by Random Forests. AdaRank follows and lastly, Ranknet. Although the NDCG scores reflected for all seem to be close to 1, the difference in performance of the algorithms is clearly observed. The reason of high NDCG scores, is probably because our data set is comparitively small with less data to train on.

Below, is a chart of performance of different algorithms for different feature combinations:





Milestones achieved

Out of the milestones, that we had stated, we have successfully managed to get the most optimum feature combination by set/unset of different feature combinations. We have also been able to carry out a comparitive study over 4 different ranking algorithms as we had stated.

References

[Masurel2013] P.Masurel,K.Lefevre-Hasegawa, C.Bourguignat, M.Scordia 2013. *Dataikus Solution to Yandexs Personalized Web Search*

[2] .J.C. Burges. From RankNet to LambdaRank to LambdaMART: An Overview. Technical Report, Microsoft Research, 2010.

[3] anklib. `http://people.cs.umass.edu/~vdang/ranklib.html`

[Gusfield1997] Dan Gusfield. 1997. *Algorithms on Strings, Trees and Sequences*. Cambridge University Press, Cambridge, UK.

[4] .Sontag et al. Probabilistic Models for Personalizing Web Search. Microsoft Research