Personalized Web Search

Dhanraj Mavilodan (dhanrajm@stanford.edu), Kapil Jaisinghani (kjaising@stanford.edu),
Radhika Bansal (radhika3@stanford.edu)

Abstract:

With the increase in the diversity of contents available on the web, the need for personalizing the search results has become more and more relevant. Usually the ranking of the pages is done with respect to query terms and page content alone. But if the ranking is done considering query terms as well as user history, then the search engine performance can be improved. In this paper, we try to learn the user interactions and aim to re-rank search query results based on the learned user interests.

1. Introduction

One criticism of search engines is that when queries are issued, most return the same results to all users. Different users may have completely different information needs and goals, when using precisely the same query again. For example, a person interested in programming may use the search query “design pattern” to find information related to software and a person in fashion industry may use the same query for the design pattern in clothes. With personalization, a search engine can find out which result to prioritize and treat the two queries differently.

2. Related Work

In the recent works like [7], KNN and K-mean based collaborative filtering is used for enriching the user history by taking into account the history of similar users. [7] shows that user profile constructed based on modified collaborative filtering achieved the best accuracy. In [1] it is shown that personalization is not effective for all queries. Like the queries with low click entropy (for which most people click on same URL) does not require personalization. For a user - query pair, give more weightage to the URL clicked by the user for that particular query. Skip click model takes hints from not just URL clicked, but also from the skipped ones. Suppose a user clicked URL at position 4, then the three URLs above it are skipped URLs. SAT-click introduced by Fox et al. [2] is measured by time between click and next action. If time is less than a certain threshold, satisfaction is 0; otherwise more satisfaction.

3. Experimental Set up

a. Dataset

Data is publicly available here as part of the personalized web search challenge on Kaggle. The dataset includes user sessions extracted from Yandex logs, with user ids, queries, query terms, URLs, their domains, URL ranking, clicks and timestamps. User data is made anonymous by using IDs instead of actuals. We have used sample from this dataset for faster experimentation and iterations of the learning algorithms. For training, we randomly picked 18856 unique users only, picked all queries by those users and then split the Kaggle provided train data into train set(first 24 days) and test set(next 3 days). We are not using the test data provided by Yandex as it does not have any click information to compute NDCG. Also we have removed those queries which does not have any click information from both train and test datasets. Table 1 & graphs in figure 1, 2 show some basic characteristics of our sampled dataset

Characteristic	Train	Test
Days	24	3
Session Count	78599	9328
Unique Users	15485	4727
Query Count	97952	13555
Unique Terms	96379	96379
Unique Domains	207080	48300
Unique URLs	799242	119461
Click Count	199076	23547
Total Records	**1590135**	**187220**

Table: 1

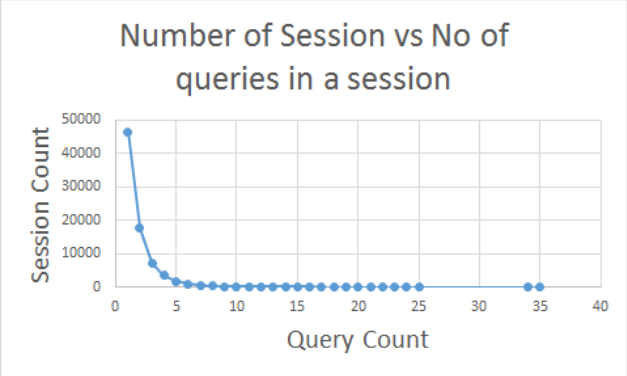

Figure: 1

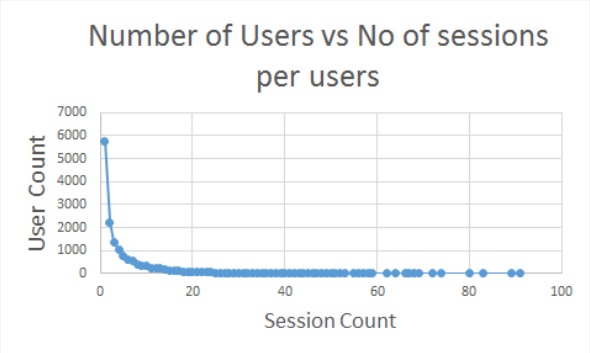

Figure 2

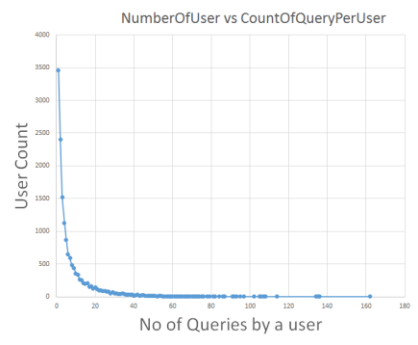

Figure 3

In figure 3, we can see that the number of users who are doing multiple queries are exponentially decreasing. So the scope of personalization is limited as the user history is also limited.

b. Evaluation Metrics

We are evaluating the results from learning algorithms using NDCG (Normalized Discounted Cumulative Gain) metric. It measures the performance of a recommendation system based on the graded relevance of the recommended entities. It varies from 0.0 to 1.0, with 1.0 representing the ideal ranking of the entities.

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

k - The maximum number of entities that can be recommended.

$IDCG_k$ is the maximum possible (ideal) DCG for a given set of queries, documents, and relevances.

4. Our Approach

The search results in the dataset, provided by Yandex, was not personalized. We worked to re-rank the search results based on the user's search history. Since we didn't have information to make explicit relevance judgment, we used implicit relevance based on clicks. Here we assume that the clicks are a proxy for what user considers relevant. And the user data provided by Yandex is anonymized by providing only IDs for sessions, users, query, URLs and their domains. So we cannot use any information on URL content, query content, user personal profile as suggested by many papers. We worked on creating features to capture personal biases based on user's search and click history and trained a ranker to re-rank the result.

4.1 Data Labelling

For a given session and query, URLs can be categorized as skipped, clicked or missed. All the URLs below the last clicked URL are missed and all the not clicked URLs above it are skipped. Here we are interested in clicked URLs. As mentioned earlier, we don't have the human judgment information to give relevance score to URLs. We use the click information and the amount of time spent on each click to label the clicked URLs into three categories.

- High Relevance : URL clicked and time spent on that is more than 300 units.
- Relevant : URL clicked and time spent is between 50 to 300 units.
- Irrelevant : URL clicked and time spent is less than 50 units or URLs missed /skipped

The amount of time spent is calculated from the timestamp information provided in the dataset. The timestamp is given for a click and query w.r.t a session and by taking the timestamp difference between the clicks, the time spent is calculated. If the click is a last activity in a session, then the time spent is given 300 units, since it can be put in satisfied category.

4.2 Normalization of Query

Since the Yandex dataset has only term IDs of a query, we couldn't apply normalization technique in queries to identify same queries. Instead we tried to find stop words using term frequency in queries. Top ten stop words are then excluded from the query terms. Also we sorted the terms to normalize queries like "interstellar movie" and "movie interstellar". This helped us to increase the correlation of similar queries from user's past history. There may be some cases where the meaning may be different, but to figure that out we need the exact term, which isn't provided in the dataset.

4.3 Data Features

This section describes the features we have derived from the given dataset. These features are used in training and testing our model.

4.3.1 Navigational query

Personalized search shouldn't be applied for navigational queries. Ex: query "facebook" indicates that people would always click the link facebook.com. To find out navigational queries, we use query entropy feature which measures the variability in clicked results across individuals. Click entropy is calculated as:

Click entropy (q) = - Σ p(C$_u$\|q) * log2(p(C$_u$\|q))

p(C$_u$|q) is the probability that URL u was clicked following query q
p(C$_u$|q) = (number of times that url is clicked for that query) / (total urls got clicked for that query)

A large click entropy means many pages were clicked (non-navigational query) for the query, while a small click entropy means only a few were clicked (navigational query).

4.3.2 User Preferences

- User's past preference on domains: Some users have preference for certain domains for a particular topic. For example some users prefer amazon while some prefer ebay for shopping. From train data, 8282 domains (out of 126448 clicked domains) got clicked by the same user more than once (Figure 4).

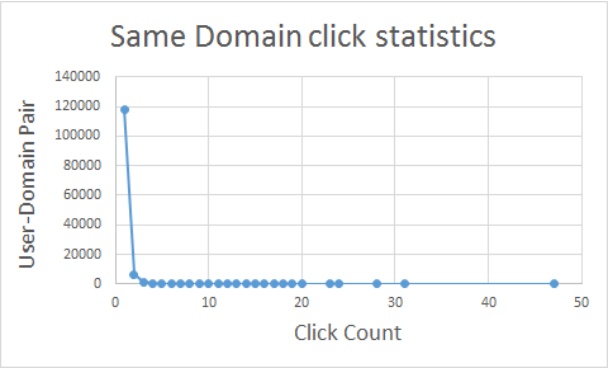

Figure 4

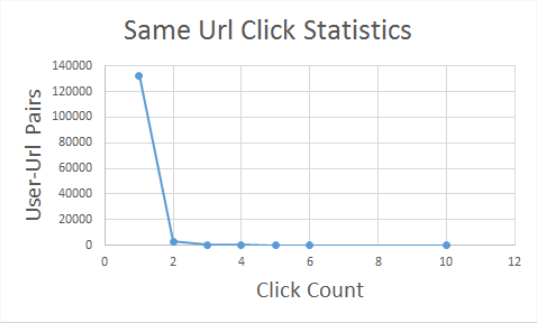

Figure 5

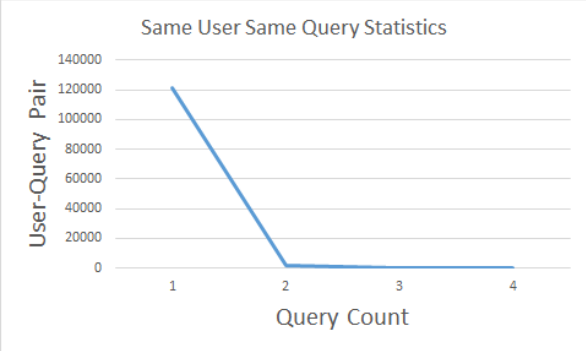

Figure 6

- User's past preference on URLs : Count for repetitive search handling (Figure 5)

  From train data, 3384 URLs (out of 135832 clicked URLs) got clicked by same user more than once.

- User's past preference on same query: User preference for a URL and domain is specific to query. For example, for tech queries user may prefer stackoverflow result, but for movie, s/he may prefer imdb. Similarly for urls, user preference may change with query type.

  From train data, 1540 queries (out of 122797) got queried by same user more than once.

For each predicate (example:-user-domain pair), calculate the following three features that correspond to three URLs categorization:

- High Relevant Prob : (Total high relevant URls for a predicate +1)/(total URLs for a predicate +3)
- Relevant Prob : (Total relevant URls for a predicate +1)/(total URLs for a predicate +3)
- Irrelevant Prob : Total irrelevant URls for a predicate +1/(total URLs for a predicate +3)

Here we used laplace smoothing to handle sparseness of data.

4.3.3 Query Based feature

URL re-ranking also depends on query ambiguity. For capturing query ambiguity, we have included two features:

- query length : smaller the length more probability of ambiguity
- query average position in session : later in the session means less ambiguous as user keeps on refining query

Feature Name	Description
User-Domain 1. userDomainHighRelevantProb 2. userDomainRelevantProb 3. userDomainIrrelevantProb	For capturing user domain preference. For a user-domain pair find out the three probabilities: High relevant, relevant, and irrelevant. Calculation is given in sec 4.3.2. Example: userDomainHighRelevantProb = (Total High Relevant URLs for a user-domain pair)/(Total URLs for a user-domain pair)
User-Domain-Query 4. userDomainQueryHighRelevantProb 5. userDomainQueryRelevantProb 6. userDomainQueryIrrelevantProb	For capturing user domain preference w.r.t to similar queries. For a user-domain-query tuple find out the three probabilities: High relevant, relevant, and irrelevant.
User-Url-Query 7. userUrlQueryHighRelevantProb 8. userUrlQueryRelevantProb 9. userUrlQueryIrrelevantProb	For capturing repeated url history for a user for similar queries. For a user-url-query tuple find out the three probabilities: High relevant, relevant, and irrelevant.
User-Url 10. userUrlHighRelevantProb 11. userUrlRelevantProb 12. userUrlIrrelevantProb	For capturing repeated url history for a user. For a user-url pair find out the three probabilities: High relevant, relevant, and irrelevant.
13. Unpersonalized rank	The rank at which the url is displayed in the original result list. It contains extremely valuable pieces of information of pagerank, query document similarity, and all other information Yandex could have used to compute its original ranking.
14. Query Entropy	For finding out navigational queries.
15. Query length	Measure for query ambiguity
16. Query average position in session	Measure for query ambiguity

Table: 2

4.4 Test Features

All the above feature extraction is done on training set based on click information. But we don't have click information on test data. To assign feature values for test data set, we use the values computed for train data where predicate value is matching.
Example: for a given user-domain pair in test data check if same pair appears in train data. If it appears, use same feature values of train data, otherwise assign the average value of that feature.

4.5 Handling Missing Feature

As mentioned in the data characteristics, most of the users searched only for a single query, which means that for a user-domain pair, there is just one URL. So there are a lot of data points where the features are not present in test data. Also in train data there are many cases where there is a single URL for a predicate value. When we didn't do any smoothing and assigned a score of one for predicates with single url, the model was giving very high performance difference between test data (0.8217 ndcg) and train data (0.98 ndcg), indicating that the model is overfitting. Also we found out that the features are biased and not giving any importance to rank. We analyzed this overfitting on the train data and handled such cases separately by giving them average score (similar to the test data) along with laplace smoothing. These changes helped us in improving ndcg on test data to 0.8238 and remove the overfitting (train data ndcg 0.88).

4.6 Re-ranking the Search Results

Re-ranking can be done by classifying the data points into one of the three classes and then ordering the URLs based on the class label. But [4] and [5] have shown that this approach is highly unstable and ordering the URLs by expected relevance is a more stable way. So we are re-ranking our urls using score function calculated using class probabilities to maximize our NDCG's expectancy. [5] has shown that NDCG's expectancy is maximized using the following equation:

$$DCG(\sigma_0, r) = \max_{\sigma \in S_n} \sum_{i=1}^{10} \frac{1}{\log_2(i+1)} \mathbf{E}\left[2^{r_{\sigma_i}} - 1 \mid X\right]$$

Maximum is obtained by sorting URLs by decreasing values of numerator:
$E\left[2^r - 1 \mid X\right]$.

That is the decreasing values of:

$$p(r = 1 \mid X) + 3p(r = 2 \mid X)$$

We modified the sort function to give a slight weightage to rank so that it will break the tie when two URLs have same class probabilities. We are giving some weightage to prob0 also to counter the cases where - URL1 has significant prob2 and prob0, but prob1 is zero; and URL2 has slightly low prob2 and prob1, but prob0 is zero. Using the original formula (decreasing order of prob1 + 3prob2), URL1 got better predicted rank even though it has significant probability of irrelevant class (prob0). So after these modifications our scoring function is:

Score = (0.3 * Log(rank)) - prob1 - (3 * prob2) + (0.5 * prob0) *(low score is better)*

- rank is non-personalized rank
- prob1 is probability of relevant class,
- prob2 is probability of high relevant class

- prob0 is probability of irrelevant class.

We then sort the URLs in increasing order of their score value so that the URL with predicted rank 1 has least score and has high importance.

5 Algorithms and Results

We have approached this problem using a point-wise approach. Point-wise algorithms typically rely on either regression or classification methods. Our approach was based on the classification of the test urls into one of the 3 classes - 0 (missed /irrelevant/skipped), 1 (relevant), 2 (highly relevant). For training our model to classify the URLs into the above classes, we have used the following algorithms:

- Random Forest
- Gradient boosted trees.

We have used the Scikit python library implementation of these algorithms.

For Random forest, we have used the following parameters:
n_estimators=10, max_depth=None, min_samples_split=2, random_state=None, n_jobs=-1

For Gradient Boosted trees, we have used the following parameters:
n_estimators=10, learning_rate=0.5, max_depth=1, random_state=0

Model	Zero Entropy Restriction (don't use personalization where query entropy is zero)	Baseline NDCG score on test data	Predicted NDCG score on train data	Baseline NDCG on train data	Predicted NDCG on train data
Gradient boosting	**No restriction**	**0.8198**	**0.8241**	**0.8181**	**0.8681**
Gradient boosting	With Restriction	0.8198	0.82261	0.8181	0.8449
Random Forest	No restriction	0.8198	0.8214	0.8181	0.8978
Random Forest	With Restriction	0.8198	0.8204	0.8181	0.8619

Table: 3

5. Conclusion and Future work.

From our results, we can see that Gradient boosting gives better performance on test data than Random Forest, for the dataset and set of features we have used. Random forest is giving more importance to predicate probability features than Yandex rank. As a result of which, it is giving better result on train data but not on test data.

From our observation, to handle this problem better, much more data needs to be processed to have a better history. For that, map-reduce approach may be used. The class labels can be increased from 3 to 4 in order to distinguish between skipped and missed clicks. Further scope of work includes adding new features like similar user preference using collaborative filtering, getting query similarities using KNN, and trying some more classification algorithms (SVM) and algorithms specific to ranking (RankNet, LambdaMART using RankLib). If unencrypted data (like actual queries instead of ID's) is available, NLP techniques can be applied to analyze query terms and page content, for better personalisation.

References

1. J. Teevan, S. T. Dumais, and D. J. Liebling. To personalize or not to personalize: Modeling queries with variation in user intent.
2. S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search
3. Milad Shokouhi, Ryen W. White, Paul Bennett, Filip Radlinski . Fighting Search Engine Amnesia:Reranking Repeated Results
4. Learning to Rank Using Classification and Gradient Boosting http://research.microsoft.com/pubs/68128/boosttreerank.pdf
5. Dataiku's Solution to Yandex's Personalized Web Search Challenge http://research.microsoft.com/en-us/um/people/nickcr/wscd2014/papers/wscdchallenge2014dataiku.pdf
6. A Large-scale Evaluation and Analysis of Personalized Search Strategies http://www2007.org/papers/paper495.pdf
7. Adaptive Web Search Based on User Profile Constructed without Any Effort from Users http://www2004.wwwconference.org/docs/1p675.pdf