Personalized Web Search
DD2476 Project Report

Martin Hwasser hwasser@kth.se Kar Shun Ip ksip@kth.se

Dmytro Kalpakchi dmytroka@kth.se Henrik Karlsson henrik10@kth.se

November 9, 2017

Abstract

This paper explains our solution to the Yandex Personalized Web Search Challenge. The goal of the challenge was to personalize search results. Our submission, using LambdaMart, achieved an NDCG@10 of 0.79638 and outperformed the original Yandex baseline.

Introduction

In 2013 Yandex organized a machine learning challenge on Kaggle called Personalized Web Search Challenge. The challenge was to re-rank documents from search engine result pages (SERP) in order to improve Normalized Discounted Cumulative Gain (NDCG) and outperform the Yandex baseline.

The challenge poses the so-called learning-to-rank problem, which applied to Information Retrieval (IR) can be stated as follows. Assume that we have a collection of documents (web pages in our case). An Information Retrieval system ranks this collection of documents with respect to a given query, assigning a score to each of them, and returns the documents in the descending order of scores. Each score reflects the relevance of the document with respect to a given query. We are interested in learning the ranking function that re-ranks the collection of documents returned by the IR system with respect to a query and a *user*, so that the documents are returned in order of descending relevance.

Related work

Learning-to-rank is a quite a popular topic in Machine Learning and Information Retrieval. According to [1] there are three major approaches to tackle this problem: pointwise, pairwise and, listwise.

Pointwise approaches try to classify each of the returned URL into one of the relevance categories. Any classification algorithm can be used for a pointwise approach.

Pairwise approaches classify pairs of returned URLs into 2 categories: correctly and incorrectly ranked. Popular pairwise approaches include RankNet [2], LambdaRank [3], IR-SVM [4].

Listwise approaches try to optimize the evaluation metrics (such as NDCG) directly averaged over all returned documents for all queries in the training data. Examples of listwise approaches include ListNet [5] and a new approach published in April 2017, ES-Rank [6].

The top three participants of the Yandex challenge were obliged to publish papers on how they tackled this specific problem. They have all used large amounts of features and trained a few learning to rank algorithms. The winners, Masurel et al [7], tried both a pointwise learning-to-rank algorithm using Random Forests, as well as the popular LambdaMart algorithm [8], the LambdaMart approach ended up winning. However, due to the difficulties of parallelizing LambdaMart and the size of the dataset, training was very slow. Similarly, the runners-up Song [9] used pointwise learning to rank model using logistic regression because the LambdaMART training went too slow. The third-place finisher Volkovs [10] also employed different learning to rank algorithms, using both neural nets and LambdaMart.

Evaluation metric

The submissions to the challenge are evaluated using NDCG (Normalized Discounted Cumulative Gain). The score of a submission is the NDCG of rankings provided by it for test queries, averaged over queries.

$$\text{NDCG@T} = \frac{\text{DCG@T}}{\text{IDCG@T}} \tag{1}$$

where

$$\text{DCG@T} = \sum_{i=1}^{T} \frac{2^{r_i} - 1}{\log_2(i+1)} \tag{2}$$

$$\text{IDCG@T} = \sum_{i=1}^{T} \frac{2^{\hat{r}_i} - 1}{\log_2(i+1)} \tag{3}$$

where $\hat{r}_i$ the relevance of document i in the ideal ordering of the documents.

In the Yandex dataset the relevance labels for the URLs are defined as the following:

Relevance	*Description*
0	Irrelevant, no clicks and clicks with dwell time $<$ 50 time units (t.u.)
1	Relevant, clicks with dwell time between 50 and 399 t.u. (inclusively)
2	Highly relevant, clicks with dwell time $\geq$ 400 t.u. and the last clicks of sessions

For documents associated with multiple clicks, the relevance grade will be determined by the click that scores the highest relevance grade according to the table above.

Data

The dataset was released in 2014 by Yandex to facilitate a research in web search personalization. The data consists of fully anonymized user sessions extracted from Yandex logs. Only meaningless numeric IDs of users, queries, query terms, sessions, URLs and their domains are released. The queries are grouped by sessions. The interesting characteristics of the dataset are presented in table 1.

Unique queries	21,073,569
Unique urls	703,484,26
Unique users	5,736,333
Training sessions	34,573,630
Test sessions	797,867
Clicks in the training data	64,693,054
Total records in the log	167,413,039

Table 1: Characterstics of Yandex dataset

Each log represents a stream of user events and can include either a query action, of type T for test data and type Q for other data, or a click action with record type C. A special event containing session metadata is also included with the type of record M.

The format of each type of logs is fully described in [11] and summarized in the table 2.

TypeOfRecord	Format
M	SessionID TypeOfRecord Day UserID
Q or T	RecordInfo QueryID ListOfTerms ListOfURLsAndDomains
C	RecordInfo URLID

Table 2: Format of Yandex dataset logs

The fields, contents or format of which needs to be clarified, are described in a more detailed way in the list below.

- RecordInfo is a record identifier consisting of 4 space-separated fields: SessionID, TimePassed, TypeOfRecord, SERPID;
- TimePassed is the time passed since the start of the session with the SessionID in units of time. The number of milliseconds in a unit of time was not disclosed by Yandex;
- ListOfTerms is a comma-separated list of terms of the query, represented by their TermIDs (each of which is unique identifier of a query term);
- SERPID is the unique identifier of a search engine result page at the session level (SERP);
- ListOfURLsAndDomains is the list of comma separated pairs of URLID and the corresponding DomainId.

Method

Sophisticated search ranking tends to involve relevance labels for search results. These labels can be manually assigned or automatically inferred. For example, Google has a big team of people manually labeling different search terms [12]. However, with anonymized data like this, an automatic approach is required. Available to us is the link the user clicked on, and how long they *dwelled* after clicking the link. We can use this dwell time to assign relevance labels, such that a low dwell time has 0 relevance while a longer dwell time have a higher relevance.

As described in the previous section, we have 30 days of historic data. The test set was comprised of days 28 to 30 while the training data comprised of day 1-27. The training data was further divided into a history set day 1-24, and a learning set day 25-27. The history set was used as the history for the training data. The training set was divided in this way to ensure that the learning set was as similar to the test set as possible.

Features

When creating features for training, we only used sessions that had at least one relevance label. This was done to reduce the amount of training data which would have otherwise been overwhelming, and to increase feature and label density.

Non-Personalized Features

Our first feature was non-personalized rank. This corresponded to the order of the search result on the original SERP. This was an important feature! We had real search results at our disposal and Yandex boasted an impressive NCDG of 0.79056 on the test set. Realizing this, we used the non-personalized rank as a proxy to whatever techniques might already be using, such as PageRank, tf-idf (or one of its variations) and document-query similarity. Furthermore, there is also an inherent bias in this type of data in that users are more likely to read top-to-bottom and thus click on the first few results rather than the last few. This bias could theoretically mean that a great personalized model will still perform worse than the baseline, because the evaluation is based on clicks for an existing model, and links after a clicked one tend to be skipped.

It contained two extremely valuable pieces of information. First, it was our only proxy to PageRank, query-document similarity, and all other information Yandex could have used to compute its original ranking.

We also decided to include the second feature connected to non-personalized rank, the exponentially decaying non-personalized rank defined as exp(−rank). The intuition behind adding this feature is the exponential decay in PageRank scores, the problem is however that the ranker can not decide the shape of the exponential decay.

Personalized Features (Set 1)

In order to enable personalization, we had features based on the user's feedback and history. There was a total of 10 features to capture personalization data and they come in pairs. They were average clicks and relevance (which is based on dwell time) under same session and query, same session and site, same session and domain, same site and similar query (the query is considered similar if they share at least one query term) in user history, and same domain and similar query in user history.

The features about same sessions captured the short-term user behavior and context while the user history-based features captured the long-term ones.

Personalized Features (Set 2)

For our best ranker, LambdaMART, we made another set of five features to see if it would improve the NDCG score. The features were: total number of clicks on the same domain or site in the same session; and total number of clicks on the same domain or site in similar queries.

This set of features was different from the first set in that the first set contained averages while this set consisted of just the amount of clicks. This set was designed to model navigational searches, that is searches that are made to find known links and these types of searches are made often and will usually have an overwhelming amount of clicks.

Technology used

Elasticsearch

Elasticsearch is a Lucene-based open-source search engine written in Java. Communication with Elasticsearch is done using a RESTful API. We used Elasticsearch to index the Yandex dataset using the Elasticsearch Python client, the total size of the indexed data was about 35GB.

Elasticsearch was then queried to retrieve training and testing sessions. To create the features for theses sessions, Elasticsearch was again queried to retrieve the user history that was then processed into features using Python.

RankLib

RankLib is a library of learning to rank algorithms for Java, part of the Lemur Project. RankLib was used to train Random Forest, LambdaRank and ListNet models used to rerank the Yandex test sessions.

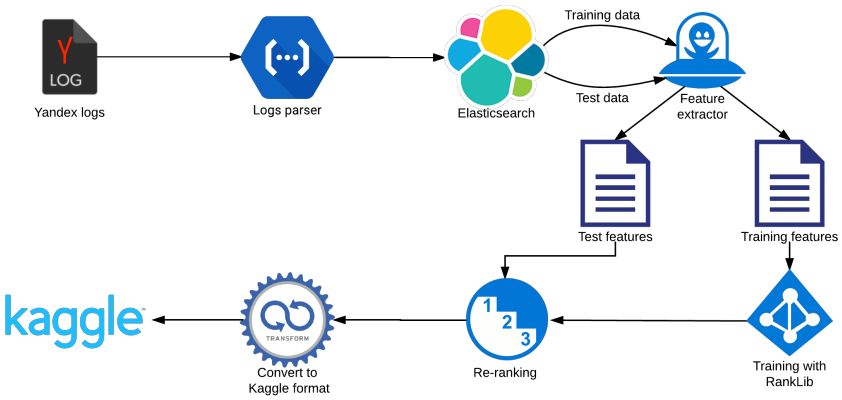

Figure 1: The workflow diagram

Learning-to-rank algorithms

LambdaMART

LambdaMART, as follows from its name, combines Multiple Additive Regression Trees (MART) and LambdaRank. MART, originally presented in [13], represents a boosted tree model that outputs the combination of multiple regression trees. One can think of all data being located in the root node of a regression tree. The objective to find such thresholds for each node so that the total variance of labels in left and right subtrees is minimized. As noted in [8], MART can then be viewed as performing gradient descent in function space using regression trees.

LambdaRank is in turn based on RankNet. These algorithms transform the problem to a pairwise classification problem. Given two URLs U_i and U_j where U_i is ranked higher than U_j, RankNet tries to minimize the cross-entropy cost function

$$C = -\bar{p}_{ij} \log p_{ij} - (1 - \bar{p}_{ij}) \log(1 - p_{ij}) \tag{4}$$

where $\bar{p}_{ij}$ is the known probability that U_i is ranked higher than U_j and p_{ij} is the learned probability. The problem is that the cross-entropy function does not directly maximize relevant information retrieval metrics.

LambdaRank addresses this issue by changing the RankNet gradient to maximize chosen metric. For example, the gradient of LambdaRank for NDCG is the RankNet gradient multiplied with $|\Delta\text{NDCG}|$ which is the change in the size of the NDCG when swapping U_i and U_j. This approach has empirically been shown to maximize the NDCG [8].

Random Forest

A pointwise approach to the learning-to-rank problem is based on predicting of either numerical or ordinal label for each returned URL. If the labels are numerical, new ranks are learned to solve regression problem, otherwise, it's a typical classification task. In our case, we have an ordinal label, the relevance which represents one of three classes (irrelevant, relevant, highly relevant).

We chose Popular Random Forests [14] as one of our rankers. Random Forest (RF) is an ensemble learning algorithm that combines multiple weak decision tree and outputs the final class label by majority vote. The main benefit of using RF over decision trees is that the risk of overfitting is lower and the variance is lower.

ListNet

ListNet is a listwise approach to ranking. A listwise approach avoids the problems which traditional pairwise ranking algorithms have, such as the loss function not directly optimizing IR measurements such as NDCG and MAP. The key to listwise approaches is a listwise loss function which makes use of probability models including permutation probability and top k probability.

ListNet [5] optimizes the listwise loss function based on top k probability. Its model is expressed using Neural Network and optimized using Gradient Descent. It claims to perform better than existing pairwise methods including Ranking SVM, RankBoost, and RankNet in learning to rank.

Results

Our best results are displayed in table 3.

In figure 2 we can see how the best model has re-ranked the search results compared to the baseline model. Relevant search results have shifted towards the first few positions, specifically toward the first, second, and third position, whereas the re-ranked model has less relevant results on positions four and five. This is important, as the first few hits are generally the most viewed ones.

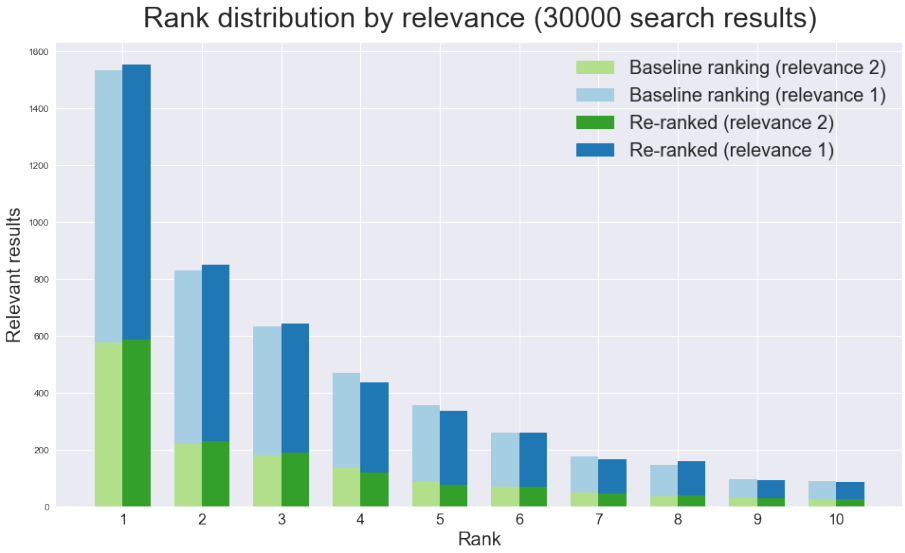

Figure 2: The distribution of search results rank. The re-ranked distribution was created by sorting by the scoring made by the LambdaMART model.

	Score		
	Training	*Validation*	*Test*
Random baseline	–	–	0.47954
Yandex baseline	–	–	0.79133
Random Forest	0.83450	0.85090	0.79572
ListNet	0.5603*	0.5939*	0.79433
LambdaMart (Feature Set 1)	0.83690	0.85300	0.79638
LambdaMart (Feature Set 1+2)	0.8463	0.8647	0.79676

Table 3: Scores for different models. All scores are NDCG@10, except ones marked with * which uses a ERR@10 scoring function.

Discussion

Our solution to the Yandex Personalized Web Search Challenge involved feature engineering for non-personalized and personalized features and multiple Learning-to-rank algorithms. It delivered promising results and gave better NDCG than the baseline solution, thereby providing a more personalized web search.

The leaderboard on Kaggle is now closed, but had it been open we would have placed in the top 50 with our best LambdaMart model. Improving on the NCDG score seemed highly plausible, but we did not have time to build more complex models due to the extensive size of the data and the slow training times of LambdaMart. The results with the LambdaMART with extra features

showed that more features gives better performance, the problem is that we don't know what features improved the performance of our LambdaMART model.

Due to a limited amount of time and computational resources we used only 17 personalization features, compared to 132 features of winning solution. Our features were different by design being focused mostly on the same session of the same user. However, even with this small set of features, we were able to outperform a baseline. Moreover, we believe that: the usage of 132 personalized features is unrealistic in terms of computational time and real world usage; the training takes to long with too many features; and that there must be redundant features.

References

[1] T.-Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends® in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009. [Online]. Available: http://dx.doi.org/10.1561/1500000016

[2] C. J. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent," August 2005. [Online]. Available: https://www.microsoft.com/en-us/research/publication/learning-to-rank-using-gradient-descent/

[3] C. J. Burges, R. Ragno, and Q. V. Le, "Learning to rank with nonsmooth cost functions," in *Advances in Neural Information Processing Systems 19*, P. B. Schölkopf, J. C. Platt, and T. Hoffman, Eds. MIT Press, 2007, pp. 193–200. [Online]. Available: http://papers.nips.cc/paper/2971-learning-to-rank-with-nonsmooth-cost-functions.pdf

[4] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon, "Adapting ranking svm to document retrieval," in *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ser. SIGIR '06. New York, NY, USA: ACM, 2006, pp. 186–193. [Online]. Available: http://doi.acm.org/10.1145/1148170.1148205

[5] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: From pairwise approach to listwise approach," in *Proceedings of the 24th International Conference on Machine Learning*, ser. ICML '07. New York, NY, USA: ACM, 2007, pp. 129–136. [Online]. Available: http://doi.acm.org/10.1145/1273496.1273513

[6] O. A. S. Ibrahim and D. Landa-Silva, "Es-rank: Evolution strategy learning to rank approach," 2017.

[7] C. B. Paul Masurel, Kenji Lefèvre-Hasegawa and M. Scordia, "Dataiku's Solution to Yandex's Personalized Web Search Challenge," 2013.

[8] C. J. Burges, "From RankNet to LambdaRank to LambdaMART: An Overview."

[9] G. Song, "Point-Wise approach for Yandex Personalized Web Search Challenge," 2014. [Online]. Available: https://www.semanticscholar.org/paper/Point-Wise-Approach-for-Yandex-Personalized-Web-Song/b838268a512bfcfa55f46d861f1257543ceb1ce5

[10] M. Volkovs, "Context Models For Web Search Personalization," *CoRR*, vol. abs/1502.00527, 2015. [Online]. Available: http://arxiv.org/abs/1502.00527

[11] "Yandex personalized web challenge on kaggle." [Online]. Available: https://www.kaggle.com/c/yandex-personalized-web-search-challenge

[12] "Google search quiality evaluation guidelines," May 2017. [Online]. Available: https://static.googleusercontent.com/media/www.google.com/en//insidesearch/howsearchworks/assets/searchqualityevaluatorguidelines.pdf

[13] J. H. Friedman, "Greedy function approximation: A gradient boosting machine." *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, 10 2001. [Online]. Available: http://dx.doi.org/10.1214/aos/1013203451

[14] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001. [Online]. Available: http://dx.doi.org/10.1023/A:1010933404324