UNIVERSITY OF AMSTERDAM

UvA-DARE (Digital Academic Repository)

Query-level Ranker Specialization

Jagerman, R.; Oosterhuis, H.; de Rijke, M.

Publication date
2017
Document Version
Final published version
Published in
Proceedings of the 1st International Workshop on LEARning Next gEneration Rankers

Link to publication

Citation for published version (APA):
Jagerman, R., Oosterhuis, H., & de Rijke, M. (2017). Query-level Ranker Specialization. In N. Ferro, C. Lucchese, M. Maistro, & R. Perego (Eds.), *Proceedings of the 1st International Workshop on LEARning Next gEneration Rankers: co-located with the 3rd ACM International Conference on the Theory of Information Retrieval (ICTIR 2017) : Amsterdam, The Netherlands, October 1, 2017* (CEUR Workshop Proceedings; Vol. 2007). CEUR-WS. http://ceur-ws.org/Vol-2007/LEARNER2017_full_2.pdf

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

UvA-DARE is a service provided by the library of the University of Amsterdam (https://dare.uva.nl)

Download date:08 Oct 2026

Query-level Ranker Specialization

Rolf Jagerman
University of Amsterdam
Amsterdam, The Netherlands
rolf.jagerman@uva.nl

Harrie Oosterhuis
University of Amsterdam
Amsterdam, The Netherlands
oosterhuis@uva.nl

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
derijke@uva.nl

ABSTRACT

Traditional Learning to Rank models optimize a single ranking function for all available queries. This assumes that all queries come from a homogenous source. Instead, it seems reasonable to assume that queries originate from heterogenous sources, where certain queries may require documents to be ranked differently. We introduce the Specialized Ranker Model which assigns queries to different rankers that become specialized on a subset of the available queries. We provide a theoretical foundation for this model starting from the listwise Plackett-Luce ranking model and derive a computationally feasible expectation-maximization procedure to infer the model's parameters. Furthermore we experiment using a noisy oracle to model the risk/reward tradeoff that exists when deciding which specialized ranker to use for unseen queries.

ACM Reference format:
Rolf Jagerman, Harrie Oosterhuis, and Maarten de Rijke. 2017. Query-level Ranker Specialization. In *Proceedings of the first International Workshop on LEARning Next gEneration Rankers, Amsterdam, October 1, 2017 (LEARNER 2017)*, 5 pages.

1 INTRODUCTION

Search engines have long been the predominant way to navigate the World Wide Web. As the amount of available content continues to grow and the popularity of search engines increases, the necessity of quality search systems is ever increasing. One of the most vital parts of any search system is the Learning to Rank (LTR) model, it considers hundreds of relevance signals and decides in what order documents should be displayed to the user [9].

Existing work has mainly focused on algorithms that optimize a single ranking model for all available queries [1, 2, 7, 9]. In contrast it seems reasonable to assume that not all queries should be ranked the same. For instance the queries *ICTIR 2017 Home Page* and *Python* require very different rankers. The former requires a ranker that focusses on retrieving a single relevant result and is thus navigational where the latter needs a ranker which provides documents relevant to different aspects of python and is more informational. In theory non-linear models could recognize this distinction and rank accordingly, however in practice the algorithms have limited capability of recognizing such query level patterns [11]. The main research question we address in this paper is:

Main RQ Can a mixture of rankers effectively recognize different ranking patterns in LTR data?

We propose a mixture of rankers approach that assigns queries to different rankers, thus these rankers become *specialized* on a subset of the available queries. Assignment is based on how the documents of a query should be ranked, thus we do not make assumptions about anything like query intent but let the model find such patterns implicitly.

2 RELATED WORK

Combining relevance signals to rank documents is the central problem in the field of learning to rank. Listwise methods have seen the greatest success [1, 2]. These approaches treat entire lists of ranked documents as learning instances. It is so successful because it can directly optimize the objective of ranking. The traditional learning to rank methods, however, learn a single ranking function or model with which all queries are ranked. In our work we capture heterogeneous ranking patterns and build a mixture of rankers based on a listwise approach.

The closest related work is the Plackett-Luce Regression Mixture Model [14]. There, the authors deal with item preferences, where users are assigned to so-called "preference groups". These groups contain users that have similar preferences in the way that they rank items. Our method bears a lot of similarity to their method, but there are two major differences that set our work apart. First, our work applies to the LTR setting, where we model queries and unique document sets for each query. Second is the fact that we deal with incomplete preferences for ranking. We consider the scenario where relevance labels are shared across documents resulting in multiple correct rankings.

3 MODEL

In this section we discuss the Specialized Rankers Model (SRM) and provide a formal definition. We contrast with existing work [14] by considering the predominant LTR scenario in Information Retrieval (IR). Here a query q_n is provided by a user at each time step and the system responds by displaying a result list of documents $\boldsymbol{r}$.

The idea of this paper is to find several specialist rankers that work very well for a subset of queries. The main assumption is that there are K query groups. Queries within a group agree with each other on how documents should be ranked whilst disagreeing with queries from other groups. Each of the query groups has a ranking function modeled by PL_k which determines the preferred order of documents. No assumptions are made about what characterizes such groups or what the reasons for their disagreement are; instead, SRM recognizes them solely based on their ranking preferences.

For this paper we assume that the ranking functions are Plackett-Luce (PL) models, thus the ranking function is a probabilistic distribution over all possible rankings of $\{d\}_n$. For brevity we will denote the feature representation of a query-document pair $\Phi(q_n, d) = \boldsymbol{d}$, the Plackett-Luce probability [10, 12] of a ranking $\boldsymbol{r}$ is then defined by:

$$PL_k(\boldsymbol{r}) = \prod_{i=1}^{|\boldsymbol{r}|} P_k(\boldsymbol{r}_i \mid \{\boldsymbol{r}_i, \boldsymbol{r}_{i+1}, ..., \boldsymbol{r}_{|\boldsymbol{r}|}\}) \tag{1}$$

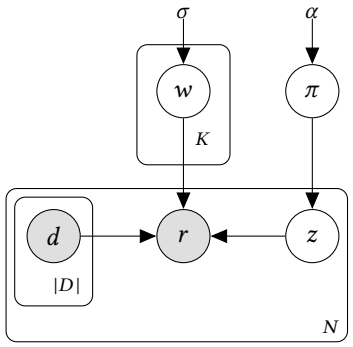

Figure 1: Plate notation of the Specialized Rankers Model, for each of the N queries there is a preferred rankings r. This ranking is determined by the ranking group indicated by the latent variable z.

where $P_k(\boldsymbol{d}_i \mid D)$ gives the probability that document d_i is sampled out of the set of remaining documents D:

$$P_k(\boldsymbol{d}_i \mid D) = \frac{\exp(f_k(\boldsymbol{d}_i))}{\sum_{d_j \in D} \exp(f_k(\boldsymbol{d}_j))}. \tag{2}$$

Here, $f_k(\boldsymbol{d})$ is a scoring function that predicts a relevance score for each document $\boldsymbol{d}$. In the field of IR the PL model is the basis for the Listwise approach introduced by Cao et al. [2]. With this model a result list is generated stepwise, where at each step a document is sampled from the remaining documents based on P_k. Since this probability depends on both the individual document as the remaining documents this approach is considered Listwise. Furthermore, in practice the PL model is only used for optimization, since non-deterministic ranking models are often avoided in production. After optimization the production ranker simply orders the documents on their $f_k(\boldsymbol{d})$ values, thus giving the ranking that maximizes the PL model [2]. The original PL model did not include a exponent and only requires every object to have a positively associated score. However, the exponent allows any scoring function f_k to be valid and is thus the prominent approach [2, 14]. For simplicity we will assume f_k is the linear model:

$$f_k(\boldsymbol{d}) = \boldsymbol{w}_k^T \boldsymbol{d}, \tag{3}$$

where the weights $\boldsymbol{w}_k$ have to be learned for each of the K ranking groups. We note, however, that any function that is differentiable with respect to its weights can be used here.

Figure 1 displays the plate notation for SRM, the generative process of the graphical model can be described as follows:

(1) The mixture proportions of the K ranking groups $\boldsymbol{\pi}$ is sampled from a Dirichlet distribution with the prior α:

$$\boldsymbol{\pi} \sim Dirichlet(\alpha). \tag{4}$$

(2) The weights $\boldsymbol{w}_k$ for each of the K ranking groups is sampled from the multivariate Gaussian distribution with zero mean and σ^2 variance:

$$\boldsymbol{w}_k \sim \mathcal{N}(\boldsymbol{0}, \sigma^2). \tag{5}$$

(3) For all queries $\{q_n \mid n = 1 \dots N\}$

(a) A ranking group z_n is sampled

$$z_n \sim \text{Multinomial}(\boldsymbol{\pi}). \tag{6}$$

(b) A ranking of its corresponding documents D_n is sampled from the PL model belonging to its ranking group z_n:

$$\boldsymbol{r}_n \sim PL(\boldsymbol{r} \mid D_n, \boldsymbol{w}_{z_n}) \tag{7}$$

Accordingly, this gives us the following likelihood:

$$\mathcal{L}(R, Z, W, \boldsymbol{\pi} \mid D) = P(\boldsymbol{\pi} \mid \alpha) \times \prod_{k=1}^{K} P(\boldsymbol{w}_k \mid \boldsymbol{0}, \sigma^2) \tag{8}$$

$$\times \prod_{n=1}^{N} P(z_n \mid \boldsymbol{\pi}) PL(\boldsymbol{r}_n \mid \boldsymbol{w}_{z_n}, D_n) \tag{9}$$

where $R = \{\boldsymbol{r}_1, \dots, \boldsymbol{r}_N\}$ is the set of all N rankings, $W = \{\boldsymbol{w}_1, \dots, \boldsymbol{w}_K\}$ the set of K weights, and $Z = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_N\}$ the set of all ranking group assignments.

The novelty of this model w.r.t. previous IR methods comes from the existence of multiple rankers which producing the preferred rankings. In contrast with the PL Regression Model [14] the SRM models queries and unique document sets for each query. Furthermore, if $K = 1$ is chosen this method reduced to the Listwise approach [2] where a single ranker produced all preferred rankings. However there is a mismatch between LTR data and the assumption that a single preferred ranking per query exists. Because most LTR datasets are based on relevance labels, some ambiguity exists since there is no preference in the order of documents with the same level of relevance. To the best of our knowledge, this ambiguity has been ignored in previous work regarding the Listwise approach, thus we contrast further by introducing a method that can handle such ambiguity.

4 OPTIMIZATION PROCESS

In order to optimize our mixture model the Expectation Maximization (EM) algorithm is used. EM is an iterative algorithm for maximizing the likelihood of a model that contains latent variables. In our case the group assignments z_n can not be observed, during optimization EM considers the expected values of the z_n and chooses the parameters $\boldsymbol{w}_k$ and $\boldsymbol{\pi}$ accordingly. By iteratively estimating z_n and then maximizing the remaining parameters, the procedure recognizes the K ranking groups while simultaneously finding the optimal rankers $\boldsymbol{w}_k$ for each group. The following sections will discuss the derivation of the expectation and maximization steps separately.

4.1 Ambiguity in LTR datasets

Unlike previous Listwise LTR work [2], we do not assume that there is just one preferred ranking $\boldsymbol{r}$. Instead we consider many different possible rankings that are correct. This generalization is appropriate because LTR datasets are usually graded on an ordinal scale, where many documents can share the same relevance label. Any permutation of the documents that corresponds to these relevance labels can be considered a correct ranking.

Each query has a corresponding set of documents $\{d\}_q$ and a set of correct rankings of these documents $\Omega_q = \{r_{q1}, r_{q2}, \dots\}$. For notational simplicity we define the probability that a correct

ranking is sampled:

$$P(\Omega_n \mid \boldsymbol{w}_{z_n}, D_n) = \sum_{r_n \in \Omega_n} PL(r_n \mid \boldsymbol{w}_{z_n}, D_n)$$

4.2 Expectation

To formulate the EM algorithm, we define the Q function as the conditional expectation of the log-likelihood:

$$\begin{aligned} Q(W, \boldsymbol{\pi}, W', \boldsymbol{\pi}') &= \mathbb{E}_{Z|R,W,\boldsymbol{\pi},D}\left[\log \mathcal{L}(R, Z, W, \boldsymbol{\pi} \mid D)\right] \\ &= \log P(\boldsymbol{\pi} \mid \alpha) + \sum_{k=1}^{K} \log P\left(\boldsymbol{w}_k \mid 0, \sigma^2\right) \\ &\quad + \sum_{n=1}^{N}\sum_{k=1}^{K} T_{nk}\left(\log P(z_n = k \mid \boldsymbol{\pi}) + \log P(\Omega_n \mid \boldsymbol{w}_k, D_n)\right). \end{aligned}$$

Thus, the output of the expectation step are the membership probabilities T_{nk}, which can be computed as follows:

$$T_{nk} = \frac{P(z_n = k \mid \boldsymbol{\pi}) P(\Omega_n \mid \boldsymbol{w}_k, D_n)}{\sum_{k'=1}^{K} P(z_n = k' \mid \boldsymbol{\pi}) P(\Omega_n \mid \boldsymbol{w}_{k'}, D_n)}. \tag{10}$$

4.3 Maximization

To maximize the Q-function, we update the model parameters $\boldsymbol{\pi}$ and W. We can compute the optimal values for $\boldsymbol{\pi}$ directly using Lagrangian optimization, obtaining the same result as [14]:

$$\boldsymbol{\pi}_k = \frac{\alpha - 1 + \sum_{n=1}^{N} T_{nk}}{\sum_{k'=1}^{K}\left(\alpha - 1 + \sum_{n=1}^{N} T_{nk'}\right)} \tag{11}$$

The optimization problem for W is more complicated. We need to find W that maximizes our Q function:

$$\begin{aligned} &\arg\max_{W} Q(W, \boldsymbol{\pi}, W', \boldsymbol{\pi}') \\ &= \arg\max_{W} \sum_{k=1}^{K}\left(\log P\left(\boldsymbol{w}_k \mid 0, \sigma^2\right) + \sum_{n=1}^{N} T_{nk} \log P(\Omega_n \mid \boldsymbol{w}_k, D_n)\right). \end{aligned}$$

Since the expression is a sum over independent terms of W, we can solve it for each $\boldsymbol{w}_k$ individually:

$$\arg\max_{\boldsymbol{w}_k} \log P\left(\boldsymbol{w}_k \mid 0, \sigma^2\right) + \sum_{n=1}^{N} T_{nk} \log P(\Omega_n \mid \boldsymbol{w}_k, D_n). \tag{12}$$

To solve this optimization problem we perform Stochastic Gradient Descent (SGD). To do this, we compute the first-order gradient of the objective function:

$$-\frac{1}{\sigma^2}\boldsymbol{w}_k + \sum_{n=1}^{N} T_{nk} \frac{\partial}{\partial \boldsymbol{w}_k} \log P(\Omega_n \mid \boldsymbol{w}_k, D_n). \tag{13}$$

This resembles a weighted and L_2-regularized SGD step, where the regularization strength is $\frac{1}{\sigma^2}$ and where each query is weighted according to T_{nk}. The major challenge that remains is computing the stochastic gradient $\frac{\partial}{\partial \boldsymbol{w}_k} \log P(\Omega_n \mid \boldsymbol{w}_k, D_n)$. Using the chain rule and product rule we end up with the following formula:

$$\sum_{r \in \Omega_n} \frac{PL(r \mid \boldsymbol{w}_k, D_n)}{P(\Omega_n \mid \boldsymbol{w}_k, D_n)} \sum_{i=1}^{|r|} \frac{\frac{\partial}{\partial \boldsymbol{w}_k} P_k(r_i \mid \{r_i, \ldots, r_{|r|}\})}{P_k(r_i \mid \{r_i, \ldots, r_{|r|}\})} \tag{14}$$

In order to compute $\frac{\partial}{\partial \boldsymbol{w}_k} P_k(r_i \mid \{r_i, \ldots, r_{|r|}\})$, we use the quotient rule and get the following result:

$$P_k(r_i \mid \{r_i, \ldots, r_{|r|}\}) \frac{\sum_{j=i}^{|x|} e^{f_k(\boldsymbol{d}_j)}\left(f_k'(\boldsymbol{d}_i) - f_k'(\boldsymbol{d}_j)\right)}{\sum_{j=i}^{|x|} e^{f_k(\boldsymbol{d}_j)}} \tag{15}$$

Plugging equation 15 into equation 14, gives us the gradient:

$$\sum_{r \in \Omega_n} \frac{PL(r \mid \boldsymbol{w}_k, D_n)}{P(\Omega_n \mid \boldsymbol{w}_k, D_n)} \sum_{i=1}^{|r|} \frac{\sum_{j=i}^{|r|} e^{f_k(\boldsymbol{d}_j)}\left(f_k'(\boldsymbol{d}_i) - f_k'(\boldsymbol{d}_j)\right)}{\sum_{j=i}^{|r|} e^{f_k(\boldsymbol{d}_j)}} \tag{16}$$

With this gradient, we can now find the optimal values for $\boldsymbol{w}_k$ using SGD. Note, however, that computing the gradient still requires computing a sum over all rankings r in Ω_n, which is prohibitively expensive for datasets with many documents per query. In the next section, we will discuss how to tackle this large computational cost.

4.4 Dealing with the complexity of ambiguity

The size of the set of correct rankings Ω_n is on the order of $O(|D_n|!)$. As a result, computing the EM steps quickly becomes infeasible when there are many documents to be ranked per query. To solve this problem we use several techniques:

Top-m. Similar to the original Listwise method [2] we use the top-m approach to reduce computational complexity. The set of correct top-m rankings is defined as:

$$\hat{\Omega}_n = \left\{\boldsymbol{r} \mid \exists_{\boldsymbol{r}' \in \Omega_n} \boldsymbol{r}_0 = \boldsymbol{r}_0' \wedge \ldots \wedge \boldsymbol{r}_m = \boldsymbol{r}_m'\right\}$$

Accordingly the likelihood that the top m documents are correct is: $P(\hat{\Omega}_n \mid \boldsymbol{w}_{z_n}, D_n)$. The derivation for the E-step and M-step remain the same, but use $\hat{\Omega}_n$ instead of Ω_n.

Dynamic programming. An important property of the PL model is that the probability of a document being sampled only depends on the set of remaining documents. Thus the order of the previously sampled documents does not matter, as a result the E-step and M-step can be computed by iterating over all subsets of documents instead of all their permutations. Using Dynamic Programming this reduces the computational complexity of our method to $O\left(2^{|D_n|}\right)$.

Importance sampling. Despite the large gains achieved with dynamic programming, there are queries where the number of relevant documents is so great that exact computations are still intractable. For these cases we have a final sampling method that approximates the gradient but can be computed in constant time. This approach samples a subset of rankings from Ω_n and then performs calculations as if this is the complete set of correct rankings. However since there is no way to directly sample Ω_n, we sample rankings from the current ranker and using importance sampling weigh them so that every ranker in Ω_n has the same weight. This approach works well because generally a small subset of Ω_n has almost all the probability mass ($PL(r \mid \boldsymbol{w}_k, D_n)$).

5 EXPERIMENTS

We evaluate our model and method on two widely used learning to rank datasets:

MSLR-WEB10k [13]. This dataset contains 10,000 queries. It represents query-document pairs as 136-dimensional feature vector

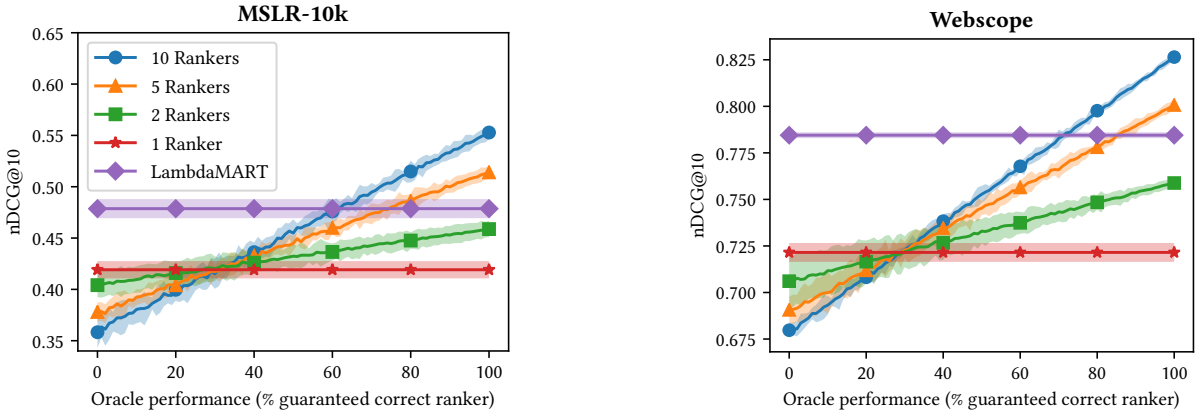

Figure 2: Ranker performance with a corrupted oracle. The X-axis indicates the oracle accuracy, where 100% means the oracle always assigns the best performing ranker to the test query and 0% means the oracle chooses a ranker uniformly at random. The shaded areas indicate a 95% confidence interval.

and grades them on a scale from 0 (irrelevant) to 4 (perfectly relevant). 6,000 queries are used for training, 2,000 for validation and 2,000 for testing.

Webscope [3]. This dataset contains 29,921 queries. The query-document pairs are represented using 519-dimensional feature vector and graded on a scale from 0 (irrelevant) to 4 (perfectly relevant). 19,944 queries are used for training, 2,994 for validation and 6,983 for testing.

We evaluate the rankings produced by our models using nDCG [6]. Given a test query, we assign one of K specialist rankers. In this paper we choose an oracle setup so that we can systematically control the number of mistakes we make in assigning a specialist ranker to a given query. An oracle's accuracy ranges from 0% (choosing a ranker uniformly at random) to 100% (choosing the best ranker every time). This allows us to study the risk/reward tradeoff associated with selecting a specialist ranker for unseen test queries.

We train the SRM model for different number of rankers ($K = 1, 2, 5, 10$) using hyperparameters $\alpha = 1.01$ and $\sigma = 0.1$. In our implementation we use Adam [8] as the SGD optimizer with a minibatch size of 128 and the default Adam settings $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 1 \times 10^{-7}$. We compare our work with LambdaMART, a state-of-the-art non-linear model [1]. The LambdaMART model's hyper-parameters (sampling rate, feature sampling rate, learning rate, maximum number of leaf nodes and minimum number of observations per leaf) are tuned on a held-out validation set.

We evaluate our model's performance on the test set using the aforementioned oracle setup. The results are shown in Figure 2. As expected, when the oracle becomes more accurate, the performance gains become greater. However, if one were to only select the best ranker less than 30% of the time, the model would perform worse than just using a single ranker. This makes sense, as selecting a suboptimal ranker for a query can have adverse effects on its ranking performance. If the oracle selects the best specialized ranker about 60% of the time, we already see a statistically significant performance gain over using a single ranker. An 80% accurate oracle outperforms LambdaMART if at least 5 to 10 specialized rankers are used.

An oracle function could in theory be realized by using query-level features. Such features occur very rarely in public LTR datasets, thus this falls outside our experimental possibilities. Previous work that uses datasets obtained from commercial search engines has shown that classifying queries, especially for specialization purposes, is possible [4, 5].

6 CONCLUSION

In this paper we introduce SRM, a model that uses a mixture of ranking functions to recognize different ranking patterns in LTR data. We describe an efficient expectation-maximization algorithm for inferring the model's parameters. Our findings show that the model picks up on latent patterns in LTR datasets.

We evaluated SRM by making use of an oracle setup, where an oracle select the best specialized ranker for the query at hand. Depending on the dataset our experiments show that a 60% accurate oracle can already significantly outperform a single linear ranker. We require an at least 80% accurate oracle to significantly outperform a state-of-the-art LambdaMART model.

There are several directions for future work: (1) In our experimental setup we use a linear function $f_k(\boldsymbol{d})$, but our model can easily be extended to non-linear functions such as deep neural networks, which are worth investigating. (2) Assigning test queries to specialized rankers based on query-level features has previously been shown to be possible [4] and would be an interesting continuation.

Acknowledgements

This research was supported by Ahold Delhaize, Amsterdam Data Science, the Bloomberg Research Grant program, the Criteo Faculty Research Award program, Elsevier, the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 312827 (VOX-Pol), the Microsoft Research Ph.D. program, the Netherlands Institute for Sound and Vision, the Netherlands Organisation for Scientific Research (NWO) under project nrs 612.-001.116, HOR-11-10, CI-14-25, 652.002.001, 612.001.551, 652.001.003, and Yandex. All content represents the opinion of the authors, which is not necessarily shared or endorsed by their respective employers and/or sponsors.

REFERENCES

[1] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[2] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *ICML*. ACM.

[3] Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge overview.. In *Yahoo! Learning to Rank Challenge*. 1–24.

[4] Xiubo Geng, Tie-Yan Liu, Tao Qin, Andrew Arnold, Hang Li, and Heung-Yeung Shum. 2008. Query dependent ranking using k-nearest neighbor. In *SIGIR*. ACM, 115–122.

[5] Ahmed Hassan, Ryen W White, and Yi-Min Wang. 2013. Toward self-correcting search engines: Using underperforming queries to improve search. In *SIGIR*. ACM, 263–272.

[6] Kalervo Järvelin and Jaana Kekäläinen. 2002. Cumulated gain-based evaluation of IR techniques. *TOIS* 20, 4 (2002), 422–446.

[7] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *SIGKDD*. ACM, 133–142.

[8] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] Tie-Yan Liu and others. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[10] R Duncan Luce. 2005. *Individual Choice Behavior: A Theoretical Analysis*. Courier Corporation.

[11] Craig Macdonald, Rodrygo LT Santos, and Iadh Ounis. 2012. On the usefulness of query features for learning to rank. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. ACM, 2559–2562.

[12] Robin L Plackett. 1975. The analysis of permutations. *Applied Statistics* 24, 2 (1975), 193–202.

[13] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. *CoRR* abs/1306.2597 (2013). http://arxiv.org/abs/1306.2597

[14] Maksim Tkachenko and Hady W Lauw. 2016. Plackett-Luce Regression Mixture Model for Heterogeneous Rankings. In *CIKM*. ACM, 237–246.