Empirical Analysis of Impact of Query-Specific Customization of nDCG: A Case-Study with Learning-to-Rank Methods

Shubhra (Santu) K Karmaker
Auburn University
Auburn, Alabama, US

Parikshit Sondhi
Snap Inc.
Mountain View, California, US

ChengXiang Zhai
University of Illinois
Urbana, Illinois, US

ABSTRACT

In most existing works, nDCG is computed for a fixed cutoff k, i.e., *nDCG@k* and some fixed discounting coefficient. Such a conventional query-independent way to compute nDCG does not accurately reflect the utility of search results perceived by an individual user and is thus non-optimal. In this paper, we conduct a case study of the impact of using query-specific nDCG on the choice of the optimal Learning-to-Rank (LETOR) methods, particularly to see whether using a query-specific nDCG would lead to a different conclusion about the relative performance of multiple LETOR methods than using the conventional query-independent nDCG would otherwise. Our initial results show that the relative ranking of LETOR methods using query-specific nDCG can be dramatically different from those using the query-independent nDCG at the individual query level, suggesting that query-specific nDCG may be useful in order to obtain more reliable conclusions in retrieval experiments.

ACM Reference Format:
Shubhra (Santu) K Karmaker, Parikshit Sondhi, and ChengXiang Zhai. 2020. Empirical Analysis of Impact of Query-Specific Customization of nDCG: A Case-Study with Learning-to-Rank Methods. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3417454

1 INTRODUCTION

nDCG is a popular measure for evaluating information retrieval systems. Given a query and a ranked list of search results, computation of *nDCG* involves summing the discounted gains of top k documents, and normalizing by the maximum possible gain that may be obtained for the query. The gain of each document is computed based on its relevance label w.r.t. the query, typically obtained from human experts or search logs, and discounted by a function of its rank. *nDCG* can be interpreted as measuring the *perceived utility of search results by a user* who does sequential browsing of the ranked list up to the rank k. The discounting coefficients account for the decrease in probability that a user will in-fact review a document at a lower rank, and the cut-off k accounts for the possibility that the user will not browse beyond position k. With this interpretation of *nDCG*, we need to make the following decisions to effectively use *nDCG*: 1) What should be the value of the cutoff k? 2) What is an optimal discounting factor for each position?

To accurately measure the perceived utility by a user for each query, the cutoff k must be set appropriately to accurately reflect where a user who uses such a query is likely to stop browsing on the ranked list. However, the current widely accepted practice for cutoff is to set k to a constant, typically aligned with the search result page (SERP) size. For example, $k = 10$ is very common. The common practice for the discounting coefficients is to use a log based discounting factor to unevenly penalize each position of the search result. The base of the log (denoted by b) is usually set to $b = 2$. Clearly, such a *query-independent* way of computing nDCG is theoretically non-optimal as it does not accurately reflect the perceived utility from a user's perspective, making such a way of using nDCG suffer from potentially very inaccurate measure of the perceived utility by an *actual user* for each query.

The non-optimality of the current practice of computing nDCG has already been noticed in multiple previous studies. For example, [8] and [14] studied variations of discounting functions and gain functions in the design of nDCG and suggested the importance of selecting an appropriate instantiation of nDCG in evaluation. However, they have not changed the common practice of using a query-independent nDCG even in highly user-sensitive domains like E-commerce search [9]. A main reason is that although the common practice of using a fixed cutoff and base-2 logarithm is theoretically non-optimal, we do not yet know empirically whether using such an inaccurate measure of nDCG would actually change any conclusion when comparing two retrieval methods or systems. This question has not been studied in any previous work and is our main research question.

To answer our research question, we conduct a case study of how the *nDCG* obtained by popular LETOR methods are affected if we vary the cutoff k as well as discounting coefficient b while computing *nDCG* and verify whether the current practice of setting $k = 10$ and $b = 2$ is justified. Specifically, we vary these two parameters of *nDCG* to obtain multiple variants of *nDCG*, which can be interpreted as representing different users, and use each variant to rank different LETOR methods. We want to see to what extent the relative ranking of different LETOR methods would be affected by the variation of those two parameters.

We experimented with eight popular LETOR methods including RankNet [2], RankBoost [6], AdaRank [15], Random Forest [1], LambdaMART [3], CoordinateAscent [10], ListNet [4] and L2 regularized Logistic Regression [5]. Experimental results show that while the relative order of these LETOR methods in terms of their obtained *nDCG* is moderately altered in the average case (average on multiple queries), the relative order can be dramatically different at the individual query level, suggesting that the current practice

CIKM '20, October 19–23, 2020, Virtual Event, Ireland

ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3417454

of using non-optimal query-independent nDCG for choosing an algorithm to be used in a search engine application has negatively impacted the utility of the search engine.

2 EXPERIMENT DESIGN

2.1 Data Set

We used the two popular LETOR datasets, i.e., "MSLR-WEB10K" [12] and "MQ2007" [11] for our experiments. The first and second dataset contains 10,000 and 1,700 queries respectively and have previously been used as benchmarks for LETOR problems [7, 13]. Within these datasets, each row corresponds to a query-document pair. The first column represents relevance label of the pair, the second column contains the query id, and the rest of columns represent features. The relevance scores are represented by an integer scale between 0 to 4 for "MSLR-WEB10K" and between 0 to 2 for "MQ2007", where 0 means non-relevant and 4(2) means highly relevant. The larger value the relevance label has, the more relevant the query-document pair is. Features corresponding to each query-document pair is represented by a n dimensional feature vector, where, $n = 136$ for dataset "MSLR-WEB10K" and $n = 46$ for dataset "MQ2007". For details on how the features were constructed, see [11] and [12].

To make the results comparable, we randomly sampled 1000 queries from both the "MSLR-WEB10K" and "MQ2007" datasets[1]. The average number of documents associated with each query was 119.06 and 41.47 for dataset "MSLR-WEB10K" and "MQ2007" respectively. We kept all the features available (136 for "MSLR-WEB10K" and 46 for "MQ2007") for all the experiments conducted in this paper.

2.2 Learning to Rank (LETOR) Methods

There are many LETOR methods proposed in the literature. Table 1 lists the popular LETOR approaches along with popular classification and regression methods that have also been used for ranking applications. For notational convenience, we assign abbreviations to each method which are used throughout the rest of the paper.

Algorithm	Short form	Algorithm	Short form
RankNet [2]	RNet	LambdaMART [3]	LMART
RankBoost [6]	RBoost	CoordinateAscent [10]	CA
AdaRank [15]	ARank	ListNet [4]	LNet
Random Forest [1]	RF	Logistic Regression [5]	L2LR

Table 1: Popular learning to rank algorithms

2.3 Research Questions

- **Q1: How Average Behavior changes with Variable Cutoff?** *How does the average nDCG@k of a method change as we vary k and how does that affect the relative performances of a group of methods? Would a different k lead to a different winner?*
  The purpose of this investigation is to see if varying k changes the average performances (in terms of *nDCG@k*) of different LETOR methods significantly. To address this question, we vary k among 5, 10, 15, 20 and 30 and report the average *nDCG@k* achieved by different LETOR methods. For each k, we also report the rank of each compared LETOR method induced by the *nDCG@k* score it achieved. Note that, for all the experiments in this paper, we report the average *nDCG@k* of five-fold cross validation experiments[2].
- **Q2: How Query-Specific Behavior changes with Variable Cutoff?** *How does variation of k changes the nDCG@k obtained for individual queries? Does it affect the nDCG@k score (for individual queries) significantly enough to alter the ranks of a group of methods being compared? For how many queries, would the conclusion on the best performing method change if we used a different k than 10? Which methods are more sensitive to the variation of k compared to other methods?*
  The purpose of this investigation is to see how the *nDCG* score for individual queries are affected if we vary k. To answer this question, we again vary k among 5, 10, 15, 20 and 30 and compute *nDCG@k* for each query and each LETOR method. Next, we asked the following three questions from these statistics: 1) For how many $< query, method >$ pairs, the performance rank of method changes as we vary k? 2) For how many queries, at least one method changed its performance rank as we vary k? 3)For how many queries, the best performer method for $k = 10$ for a particular query could not achieve the best performance for some $k > 10$ for the same query? We also examine for how many queries, each method changes their performance rank as we vary k and compute the standard deviation of the rank values it achieved for different k's. This would allow us to examine which methods are more sensitive to the variation of k as well as which are more robust.
  Next, we look at how the best performer distribution changes with variation of k. For a particular k, we compute the number of queries for which each method obtained the highest *nDCG@k*. We then fix k to a different number and compute the same best performer counts again. This way, we examine how the best performance counts for each method change as we vary k. This also provides a way to examine which k is optimal for which method.
- **Q3: How Average Behavior changes with Variable Discounting Coefficients?** Here, we conduct similar average behavior experiments as for Variable Cutoff (research question 1), except that, instead of varying the cut-off k, we fix $k = 10$ and vary the discounting coefficient b among 2, 3, 4, 5 and 6 and finally, report the average *nDCG* scores.
- **Q4: How Query-specific Behavior changes with Variable Discounting Coefficients?** Here, we conduct similar query-specific behavior experiments as for Variable Cutoff (research question 2), except that, instead of varying the cut-off k, we fix $k = 10$ and vary the discounting coefficient b among 2, 3, 4, 5 and 6 and finally, report the query-specific *nDCG* scores.

3 EXPERIMENT RESULTS

Q1 [Average Behavior with Variable Cutoff]: Table 2 and 3 summarizes the average performances of different LETOR methods for different values of k, i.e., $k = \{5, 10, 15, 20, 30\}$ for "MSLR-WEB10K" and "MQ2007" dataset respectively. First of all, for $k = 10$ which is the widely accepted practice, LambdaMART achieves the best performance for 'MSLR-WEB10K" dataset with an *nDCG*@10 score of 0.4298, while Random Forest turns out to be the best for "MQ2007" dataset with *nDCG*@10 equal to 0.4291. On the other hand, the variance in the performance by different LETOR methods is way higher for "MSLR-WEB10K" dataset than "MQ2007".

[1] The exact datasets used for our the experiments can be found at the following link: https://bitbucket.org/karmake2/practicalndcg/downloads/
[2] All the codes and evaluation scripts for experimentation can be found at the following link: https://bitbucket.org/karmake2/practicalndcg/src

Method	nDCG@				
	5	**10**	**15**	**20**	**30**
ARank	0.3084 (5)	0.3377 (5)	0.3609 (5)	0.3807 (5)	0.4176 (5)
LNet	0.1529 (8)	0.1810 (8)	0.2044 (8)	0.2257 (8)	0.2658 (8)
RBoost	0.3164 (4)	0.3422 (4)	0.3630 (4)	0.3825 (4)	0.4185 (4)
RF	0.3860 (3)	0.4133 (3)	0.4338 (2)	0.4481 (2)	0.4777 (2)
RNet	0.1635 (7)	0.1924 (7)	0.2166 (7)	0.2360 (7)	0.2732 (7)
CA	0.3996 (2)	0.4150 (2)	0.4311 (3)	0.4443 (3)	0.4713 (3)
L2LR	0.2037 (6)	0.2431 (6)	0.2760 (6)	0.3023 (6)	0.3493 (6)
LMART	**0.4058** (1)	**0.4298** (1)	**0.4449** (1)	**0.4598** (1)	**0.4892** (1)

Table 2: Relative performances of different LETOR methods for variable k: Dataset "MSLR-WEB10K"

Method	nDCG@				
	5	**10**	**15**	**20**	**30**
ARank	0.3881 (4)	0.4156 (5)	0.4480 (5)	0.4797 (5)	0.5372 (5)
LNet	0.3754 (8)	0.4040 (8)	0.4347 (8)	0.4673 (8)	0.5272 (8)
RBoost	0.3834 (6)	0.4140 (6)	0.4490 (4)	0.4807 (4)	0.5355 (6)
RF	**0.4020** (1)	**0.4291** (1)	**0.4590** (1)	**0.4907** (1)	**0.5470** (1)
RNet	0.3794 (7)	0.4118 (7)	0.4449 (7)	0.4751 (7)	0.5344 (7)
CA	0.3967 (2)	0.4261 (2)	0.4585 (2)	0.4880 (3)	0.5446 (2)
L2LR	0.3873 (5)	0.4159 (4)	0.4474 (6)	0.4780 (6)	0.5380 (4)
LMART	0.3913 (3)	0.4213 (3)	0.4562 (3)	0.4883 (2)	0.5432 (3)

Table 3: Relative performances of different LETOR methods for variable k: Dataset "MQ2007"

For both the datasets and for each method, average *nDCG@k* obtained by the method increases as we increase k. For example, LambdaMART achieves *nDCG* value of 0.4298 and 0.4892 for $k = 10$ and $k = 30$ respectively for "MSLR-WEB10K" dataset. This is intuitive because the increase in ideal DCG is usually smaller than the same for DCG obtained by different methods as we increase k. This is likely because the ideal DCG is based on the perfect ranking and all the relevant documents are assumed to be ranked at the top positions sorted by their relevance. Thus, ideal DCG has little room to increase over DCG obtained by different methods as k becomes larger and thus, we see an increasing trend in *nDCG@k*.

Finally, we report the ranks (in brackets) of different LETOR methods induced by their *nDCG@k* scores for different k's. Results from Table 2 and 3 show that, performance rank change for a LETOR method corresponding to varying k is more common in case of "MQ2007" dataset than "MSLR-WEB10K". For example, AdaRank, RankBoost, Coordinate Ascent, L2LR and LambdaMART all changed their performance ranks at least once as k was varied for "MQ2007" dataset (Table 3). While, for "MSLR-WEB10K" dataset, only Coordinate Ascent and Random Forest switched their ranks (Table 2). This means that, the common practice of setting $k = 10$ while computing NDCG@k for comparing a set of LETOR methods is not always optimal and setting k to a different value can lead to a different conclusion regarding their relative performances.

Q2 [Query-Specific Behavior with Variable Cutoff]: In comparison to the average behavior, the *Query-Specific Behavior* we observed is even more interesting. Table 4 presents a brief summary of the *Query-Specific Behavior* with respect to variable k. We experimented with 8 LETOR methods as before (Listed in table 1) and used exactly the same pool of 1000 queries from both "MSLR-WEB10K" and "MQ2007" datasets used for *Average Behavior* analysis. Thus, there are 8000 <query, method> pairs in our experiment for each dataset and out of these 8000 pairs, we found 5969 and 5641 pairs from "MSLR-WEB10K" and "MQ2007" dataset respectively where the *method* changed its rank (induced by the *nDCG@k* score it obtained) as we varied k among $\{5, 10, 15, 20, 30\}$. This is a very

Statistic	**WEB10K**	**MQ2007**
Total # of Rank Change	5969/8000	5641/8000
# of Queries with at least one change of rank	959/1000	813/1000
# of Times the best performer is switched	449/1000	432/1000

Table 4: Summary of rank changes by different LETOR methods for individual queries with variable k

strong indication that *nDCG@k* obtained by any particular LETOR method for individual queries is highly sensitive to the variation of k. We also look at two more important statistics: first, out of the 1000 queries we considered, we look at the number of queries for which at least one of the LETOR methods changed its rank as we varied k. To our surprise, we found out the number to be 959 ("MSLR-WEB10K") and 813 ("MQ2007"). Being motivated by this, we looked for a stronger case where we counted the number of queries for which the best performing LETOR method for $k = 10$ no longer remains the best performing one at least for some $k > 10$. Surprisingly, this number was also very high, i.e, 449 ("MSLR-WEB10K") and 432 ("MQ2007"). This means for almost half of the total queries, the best performing LETOR method for $k = 10$ is outperformed by some other method when k was set to some value greater than 10. All these results strongly suggest that *nDCG@k* obtained by different LETOR methods for individual queries are indeed sensitive to the variation of k and thus, query level customization of *nDCG* computation should be done for performance evaluation of the search systems.

To further investigate the *Query-Specific Behavior* of *nDCG@k* for variable k, we look at statistics for individual methods. Table 5 presents these results. It reports the number of queries (out of 1000 queries) for which each LETOR method changed their rank as we varied k among $\{5, 10, 15, 20, 30\}$. In consistence with the previous results, we found these numbers to be very high also for both datasets . For example, in case of "MSLR-WEB10K" dataset, *Random Forest* switched its rank for 812 queries as we varied k. We also report the standard deviation in the ranks achieved by each method in Table 5. These results again demonstrate that varying k has a huge impact on *nDCG@k* obtained for an individual query.

Finally, to better analyze the *Query-Specific Behavior* of individual methods, we report the number of queries for which each method turned out to be the best performer. We report these best performance counts by individual methods with variation of k in Table 6 for both datasets. For example, Table 6 reports that *AdaRank* turned out to be the best performer for 161 queries (out of 1000 queries) for $k = 5$ (MSLR-WEB10K dataset). However, for $k = 15$, it achieved best performance for only 126 queries. The same numbers obtained by LambdaMART were 127 ($k = 5$) and 180 ($k = 15$) for MQ2007 dataset.

Q3 [Average Behavior with Variable Discounting Coefficients]: We found some evidence to the fact that average behavior is sensitive to small variations in the Discounting Coefficients. For example, consistently with the results of *Variable Cut-off* in case of "MSLR-WEB10K" dataset (Table 2), we observe that Coordinate Ascent and Random Forest switched their ranks as b is increased from 2 (Table 7). However, in contrast with the results of *Variable Cut-off* on "MQ2007" dataset (Table 3), this time only AdaRank and RankBoost switched their ranks (Table 8) [further details omitted due to lack of space].

Method	MSLR-WEB10K		MQ2007	
	Rank Switch	Rank STD	Rank Switch	Rank STD
RF	812/1000	0.6381	696/1000	0.7184
CA	801/1000	0.6344	715/1000	0.8174
RBoost	794/1000	0.6481	727/1000	0.7267
ARank	772/1000	0.6159	719/1000	0.8026
L2LR	716/1000	0.6126	669/1000	0.7218
LMART	708/1000	0.5865	663/1000	0.7434
RNet	693/1000	0.5209	733/1000	0.7342
LNet	665/1000	0.4807	719/1000	0.7066

Table 5: Statistics of rank changes by different LETOR methods for individual queries with variable k

Method	MQ2007 $k = \{5, 10, 15, 20, 30\}$	MSLR-WEB10K $k = \{5, 10, 15, 20, 30\}$
ARank	422, 317, 291, 277, 253	161, 138, 126, 123, 123
LNet	45, 48, 47, 53, 52	17, 20, 22, 18, 19
RBoost	75, 94, 100, 96, 98	125, 109, 91, 105, 98
RF	121, 135, 139, 153, 163	171, 180, 200, 215, 197
RNet	78, 71, 59, 56, 54	52, 45, 42, 38, 36
CA	35, 65, 68, 68, 75	155, 181, 187, 173, 174
L2LR	97, 114, 116, 114, 124	59, 55, 61, 60, 66
LMART	127, 156, 180, 183, 181	260, 272, 271, 268, 287

Table 6: Best performance counts by different LETOR methods at individual query level with variable k

Method	nDCG@10 for b=				
	2	3	4	5	6
ARank	0.3377 (5)	0.3501 (5)	0.3562 (5)	0.3600 (5)	0.3626 (5)
LNet	0.1810 (8)	0.1923 (8)	0.1979 (8)	0.2013 (8)	0.2037 (8)
RBoost	0.3422 (4)	0.3545 (4)	0.3606 (4)	0.3643 (4)	0.3668 (4)
RF	0.4133 (3)	0.4256 (2)	0.4319 (2)	0.4358 (2)	0.4385 (2)
RNet	0.1924 (7)	0.2042 (7)	0.2100 (7)	0.2136 (7)	0.2160 (7)
CA	0.4150 (2)	0.4208 (3)	0.4243 (3)	0.4265 (3)	0.4281 (3)
L2LR	0.2431 (6)	0.2586 (6)	0.2662 (6)	0.2709 (6)	0.2742 (6)
LMART	0.4297 (1)	0.4402 (1)	0.4456 (1)	0.4490 (1)	0.4513 (1)

Table 7: Relative performances ($nDCG@10$) of different LETOR methods for varying b: MSLR-WEB10K dataset.

Method	nDCG@10 for b=				
	2	3	4	5	6
ARank	0.4156 (5)	0.4288 (5)	0.4360 (5)	0.4407 (6)	0.4440 (6)
LNet	0.4040 (8)	0.4180 (8)	0.4255 (8)	0.4304 (8)	0.4338 (8)
RBoost	0.4141 (6)	0.4282 (6)	0.4360 (6)	0.4409 (5)	0.4444 (5)
RF	0.4291 (1)	0.4426 (1)	0.4499 (1)	0.4546 (1)	0.4579 (1)
RNet	0.4119 (7)	0.4259 (7)	0.4336 (7)	0.4386 (7)	0.4420 (7)
CA	0.4261 (2)	0.4400 (2)	0.4475 (2)	0.4524 (2)	0.4558 (2)
L2LR	0.4159 (4)	0.4302 (4)	0.4378 (4)	0.4427 (4)	0.4461 (4)
LMART	0.4213 (3)	0.4344 (3)	0.4417 (3)	0.4465 (3)	0.4499 (3)

Table 8: Relative performances ($nDCG@10$) of different LETOR methods for varying b: MQ2007 dataset.

Statistic	MSLR-WEB10K	MQ2007
Total # of Methods Changed their rank	2457/8000	1674/8000
# of Queries with at least one change of rank	701/1000	420/1000
# of Times the best performer is switched	253/1000	126/1000

Table 9: Summary of rank changes by different LETOR methods for individual queries with variable b

Q4 [Query-Specific Behavior with Variable Discounting Coefficients] We found that the query level *nDCG* is indeed quite sensitive to the variation of the discounting factor b (Table 9, 10 and 11) and should be accounted for. For example, table 11 reports that *Coordinate Ascent* turned out to be the best performer for 181 queries (out of 1000 queries) for $b = 2$ in case of MSLR-WEB10K. However, for $b = 6$, it achieved best performance for only 167 queries. The same numbers obtained by Random Forest were 135 ($b = 2$) and 146 ($b = 6$) for MQ2007 dataset [further details omitted due to lack of space].

Method	MSLR-WEB10K		MQ2007	
	Rank Switch	Rank STD	Rank Switch	Rank STD
CA	368/1000	0.2057	214/1000	0.1316
RF	358/1000	0.2015	200/1000	0.1289
RBoost	354/1000	0.2130	214/1000	0.1347
LMART	350/1000	0.2012	229/1000	0.1658
ARank	341/1000	0.1997	213/1000	0.1342
L2LR	272/1000	0.1585	201/1000	0.1253
RNet	225/1000	0.1264	199/1000	0.1230
LNet	189/1000	0.1050	204/1000	0.1224

Table 10: Statistics of rank changes by different LETOR methods for individual queries with variable b

Method	MQ2007 $b = \{2, 3, 4, 5, 6\}$	MSLR-WEB10K $b = \{2, 3, 4, 5, 6\}$
ARank	317, 320, 320, 319, 319	138, 140,143,144,145
LNet	48, 46, 45, 47, 48	20, 20, 20, 20, 20
RBoost	94, 95, 88, 90, 90	109 , 107, 106, 105, 108
RF	135, 145, 149, 149, 146	180, 186, 190, 200, 201
RNet	71, 71, 70, 67, 67	45, 43, 45, 43, 44
CA	65, 67, 67, 64, 64	181, 171, 171, 171, 167
L2LR	114, 109, 112, 114, 115	55, 52, 53, 54, 55
LMART	156, 147, 149, 150, 151	272, 281, 272, 263, 260

Table 11: Counts of best performance achieved by different LETOR methods at individual query level with variable b

In summary, an overall conclusion we can draw through our study is that the current common practice of using a query-independent fixed cutoff and query-independent discounting coefficients is problematic and query-specific nDCG may be useful in order to obtain more reliable conclusions in retrieval experiments.

REFERENCES

[1] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5–32.

[2] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the 22nd international conference on Machine learning*. ACM, 89–96.

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

[4] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*. ACM, 129–136.

[5] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A library for large linear classification. *The Journal of Machine Learning Research* 9 (2008), 1871–1874.

[6] Yoav Freund, Raj Iyer, Robert E Schapire, and Yoram Singer. 2003. An efficient boosting algorithm for combining preferences. *The Journal of machine learning research* 4 (2003), 933–969.

[7] Yasser Ganjisaffar, Rich Caruana, and Cristina Videira Lopes. 2011. Bagging gradient-boosted trees for high precision, low variance ranking models. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*. ACM, 85–94.

[8] Evangelos Kanoulas and Javed A Aslam. 2009. Empirical justification of the gain and discount function for nDCG. In *Proceedings of the 18th ACM conference on Information and knowledge management*. ACM, 611–620.

[9] Shubhra Kanti Karmaker Santu, Parikshit Sondhi, and ChengXiang Zhai. 2017. On application of learning to rank for e-commerce search. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 475–484.

[10] Donald Metzler and W Bruce Croft. 2007. Linear feature-based models for information retrieval. *Information Retrieval* 10, 3 (2007), 257–274.

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

[12] Tao Qin and Tie-Yan Liu. 2010 (Accessed: 2018-01-20). *Microsoft Learning to Rank Datasets*. https://www.microsoft.com/en-us/research/project/mslr/

[13] Shilpa Shukla, Matthew Lease, and Ambuj Tewari. 2012. Parallelizing ListNet training using spark. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1127–1128.

[14] Ellen M Voorhees. 2001. Evaluation by highly relevant documents. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 74–82.

[15] Jun Xu and Hang Li. 2007. Adarank: a boosting algorithm for information retrieval. In *ACM SIGIR*. ACM, 391–398.