An apple-to-apple comparison of Learning-to-rank algorithms in terms of Normalized Discounted Cumulative Gain

Róbert Busa-Fekete[1 2] and **György Szarvas**[3] and **Tamás Éltető** [4] and **Balázs Kégl** [5]

Abstract. The Normalized Discounted Cumulative Gain (NDCG) is a widely used evaluation metric for learning-to-rank (LTR) systems. NDCG is designed for ranking tasks with more than one relevance levels. There are many freely available, open source tools for computing the NDCG score for a ranked result list. Even though the definition of NDCG is unambiguous, the various tools can produce different scores for ranked lists with certain properties, deteriorating the empirical tests in many published papers and thereby making the comparison of empirical results published in different studies difficult to compare. In this study, first, we identify the major differences between the various publicly available NDCG evaluation tools. Second, based on a set of comparative experiments using a common benchmark dataset in LTR research and 6 different LTR algorithms, we demonstrate how these differences affect the overall performance of different algorithms and the final scores that are used to compare different systems.

1 Introduction

In *subset ranking* [3] (or *web page ranking*), the goal is to learn a ranking function that approximates the ideal partial ordering of a set of objects (or documents retrieved for the same query). The partial ordering is provided by relevance labels representing the relevance of documents with respect to the query on an absolute scale.

In the past, manually designed ranking functions, such as BM25 [7], were used to rank the retrieved documents in web page ranking. More recently, this problem is tackled as a machine learning task, where the training data is given in the form of (query, document, relevance label) triplets. These machine learning based ranking approaches are referred to as *learning-to-rank* (LTR) systems.

The Normalized Discounted Cumulative Gain (NDCG) is a widely used evaluation metric for learning-to-rank systems. NDCG is designed for ranking tasks with more than one relevance levels. There are many freely available, open source tools for computing the NDCG score for a ranked result list (for full list of these tools see Section 3). Even though the definition of NDCG is unambiguous[6], the various tools can produce different scores for ranked lists with certain properties, deteriorating the empirical tests in many published papers and thereby making the comparison of empirical results published in different studies difficult to compare.

We found that for certain benchmark datasets, the relative order of the performance of different LTR methods can change depending on which evaluation tool was used. The reason for this can be two-fold. First, the implemented NDCG calculation can itself result in different scores, and in some cases, a different order for the same ranked result lists. Second and more importantly, we found some of the LTR algorithms that compute the NDCG scores during the training phase to be sensitive to the different ways of computing the NDCG score, i.e. depending on which NDCG implementation is used, some LTR methods can produce different models that provide significantly different overall performance. In previous work[2], we recognized this, here we investigate the effect of using different evaluation tools more detailed.

The contribution of this study is two-fold. First, we identify the major differences between the various publicly available NDCG evaluation tools. Second, based on a set of comparative experiments using a common benchmark dataset in LTR research and 6 different LTR algorithms, we demonstrate how these differences affect the overall performance of different learning methods and the final scores that are used to compare different systems. We analyze these differences and also draw conclusions on which metric should be used and why.

This is not only an important step towards making the empirical research results reported in the literature uniformly comparable, but it also has implications on how benchmark datasets should be designed in order to be equally suited to train and evaluate any particular learning to rank algorithm.

The rest of this paper is organized as follows. In the next section we will describe the formal setup, and then, in Section 3 we list all the evaluation tools that are available freely to compute NDCG and, in addition, we identify precisely the differences in the way they compute the NDCG score. Next, in Section 4 we briefly overview the LTR algorithms we are used for comparison in the experiments in Section 5. Finally, based on the observed differences in experimental results, we draw our conclusions in Section 6.

[1] Department of Mathematics and Computer Science, Marburg University, Germany, email: busarobi@mathematik.uni-marburg.de

[2] R. Busa-Fekete is on leave from the Research Group on Artificial Intelligence of the Hungarian Academy of Sciences and University of Szeged.

[3] Computer Science Department Technische Universität Darmstadt, Germany, email :szarvas@tk.informatik.tu-darmstadt.de

[4] Ericsson Hungary, Könyves Kálmán krt. 11.B, 1097 Budapest, Hungary, email: tamas.elteto@ericsson.com

[5] Linear Accelerator Laboratory (LAL) and Computer Science Laboratory (LRI), University of Paris-Sud, CNRS, Orsay, 91898, France, email: balazs.kegl@gmail.com

[6] See, for example `http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-ranked-retrieval-results-1.html`

2 Formal LTR task

In this section we formally define the learning-to-rank problem and introduce the notation that will be used in the rest of the paper. Let us assume that we are given a set of *query objects* $\mathbf{D} = \{\mathcal{D}^{(1)}, \ldots, \mathcal{D}^{(M)}\}$. Each query object $\mathcal{D}^{(k)}$ consists of a set of $n^{(k)}$ pairs

$$\mathcal{D}^{(k)} = \Big\{ \big(\mathbf{x}_1^{(k)}, \ell_1^{(k)}\big), \ldots, \big(\mathbf{x}_{n^{(k)}}^{(k)}, \ell_{n^{(k)}}^{(k)}\big) \Big\}.$$

The real-valued feature vectors $\mathbf{x}_i^{(k)}$ represent the kth query and the ith document received as a potential hit for the query.[7] The *label index* $\ell_i^{(k)}$ of the query-document pair $\mathbf{x}_i^{(k)}$ is an integer between 1 and K. We assume that we are given a set of numerical *relevance grades*

$$\mathcal{Z} = \{z_1, \ldots, z_K\}.$$

The relevance grade $z_i^{(k)} = z_{\ell_i^{(k)}}$ expresses the relevance of the ith document to the kth query on a numerical scale. A popular choice for the numerical relevance grades is

$$z_\ell = 2^{\ell - 1} - 1 \tag{1}$$

for all $\ell = 1, \ldots, K$.

The goal of the ranker is to output a permutation $\mathbf{j}^{(k)} = (j_1, \ldots, j_{n^{(k)}})$ over the integers $(1, \ldots, n^{(k)})$ for each query object $\mathcal{D}^{(k)}$. A widely used empirical measure of the quality of the permutation $\mathbf{j}^{(k)}$ is the Discounted Cumulative Gain (DCG)

$$\mathrm{DCG}\big(\mathbf{j}^{(k)}, \mathcal{D}^{(k)}\big) = \sum_{i=1}^{n^{(k)}} c_i z_{j_i}^{(k)}, \tag{2}$$

where c_i is the *discount factor* of the i^{th} document in the permutation. The most common discount factor is

$$c_i = \frac{1}{\log(1+i)}. \tag{3}$$

The rationale of this formula is that a user will be particularly satisfied if he/she finds relevant documents early in the permutation. To normalize DCG between 0 and 1, (2) is usually divided with the DCG score of the best permutation (NDCG) which can be computed as

$$\mathrm{IDCG}\big(\mathcal{D}^{(k)}\big) = \max_{\mathbf{j}} \mathrm{DCG}\big(\mathbf{j}, \mathcal{D}^{(k)}\big)$$

This score referred to as the *ideal* DCG score. It is also a common practice to truncate the sum (2) at $n_{\max}$, defining the $\mathrm{DCG}_{n_{\max}}$ and $\mathrm{NDCG}_{n_{\max}}$ scores. The reason for this is that a user would rarely browse beyond the first page of search results containing the first $n_{\max}$ hits.

3 Evaluation tools

In this subsection we briefly describe and compare the various tools available to compute NDCG scores. The definition (2) is unambiguous, nevertheless, the tools can differ in the definition of the discount factor c_i (3). More importantly, there can be an important difference in the way the DCG score is normalized when i) there is no relevant document for a query ($z_i = 0$ for all i), or ii) when the number of documents is less then the truncation level $n_{\max}$. Although this seems to be a technical subtlety, it turns out that the confusion between the different tools can significantly alter the numerical scores and in some case can even change the relative ordering of the algorithms on benchmark datasets.

We compared six evaluation tools computing the NDCG scores:

1. The LETOR 3.0 script implemented in Perl[8]
2. The LETOR 4.0 script implemented in Perl[9]
3. The MS script implemented in Perl[10]
4. The YAHOO script implemented in Python[11]
5. The RANKLIB package implemented in Java[12]
6. The TREC evaluation tool v8.1 implemented in C[13]

The evaluation tools can be divided into three groups. The tools of the first group compute $\mathrm{DCG}_{n_{\max}}$ according to the definition (2) described in Section 2. The LETOR 3.0, YAHOO, and TREC tools belong to this group. All of these tools assign zero score to a query if it is *empty*, that is, $z_i = 0$ for all i which means that there are no relevant documents. The TREC tool uses the labels of documents given in the input file as relevance grades by default. From this point of view, this is the most flexible implementation, since arbitrary relevance grades can be defined. For example, in the case of MQ2008 dataset, the labels 0, 1 and 2 should be simply replaced by 0, 1 and 3 respectively, to have the commonly used exponential grades (1).

The second group is composed of the YAHOO tool alone. It also computes the $\mathrm{DCG}_{n_{\max}}$ according to the definition (2), but it assigns 1.0 to the empty queries. This is a minor difference that generates an additive bias between the $\mathrm{NDCG}_{n_{\max}}$ computed by YAHOO tool and the three tools of the first group.

The third group consists of the LETOR 4.0 and MS tools. Except for a small technical difference (the LETOR 4.0 tool can be used for up to three relevance labels, whereas the MS tool can handle up to five relevance labels), they compute the same score. As the RANKLIB and LETOR 3.0 tools, they assign zero to a query where the ideal $\mathrm{DCG}_{n_{\max}}$ is zero. Their rather strange feature is that they also assign zero $\mathrm{DCG}_{n_{\max}}$ score to a query with less then $n_{\max}$ documents in it, even if these documents are highly relevant. So, formally, they compute the $\mathrm{DCG}_{n_{\max}}$ score as

$$\mathrm{DCG}_{n_{\max}}\big(\mathbf{j}^{(k)}, \mathcal{D}^{(k)}\big) = \begin{cases} \sum_{i=1}^{n_{\max}} c_i z_{j_i}^{(k)} & \text{if } n_{\max} \le n^{(k)} \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

This truncation does not only distort the *test* score, but it can also alter the *training* of such algorithms that depend directly on the NDCG score. Indeed, for example, in ADARANK [9] which optimizes the NDCG_{10} evaluation metric, a query containing less than 10 documents does not influence the computation of the coefficient of the weak ranker at all, and the the weight of such queries converge to zero over the boosting iterations.

The only source of difference between the tools that have not been identified yet, is the way they sort the objects of interest based on the scores. All tools except TREC tool, make use of the default built-in, programing language dependent sorting function which mainly

[7] When it is not confusing, we will omit the query index and simply write $\mathbf{x}_i$ for the ith document of a given query.

[8] http://research.microsoft.com/en-us/um/beijing/projects/letor/LETOR3.0/EvaluationTool.zip

[9] http://research.microsoft.com/en-us/um/beijing/projects/letor/LETOR4.0/Evaluation/Eval-Score-4.0.pl.txt

[10] http://research.microsoft.com/en-us/projects/mslr/eval-score-mslr.pl.txt

[11] http://learningtorankchallenge.yahoo.com/evaluate.py.txt

[12] http://www.cs.umass.edu/~vdang/ranklib.html

[13] http://trec.nist.gov/trec_eval/

implement the quick sort algorithm. Therefore, the order of two elements with the same score depends on the implementation of the sorting algorithm used. Whereas the TREC tool uses the lexicographic ordering based on document ID given in the input file if the system-predicted scores are equal for two documents.

We believe that none of these two ways of handling equal scores are desirable. On the one hand, the sorting algorithm should not have an effect on the evaluation itself. On the other hand, the document ID normally does not bear any useful information regarding the content of a document, so in this sense, the use of the ordering based on document ID corresponds to a particular random ordering which is also not a desired feature in an evaluation tool.

4 Learning-to-rank algorithms

In our comparison study, we used six state-of-the-art ranking methods. Here we briefly summarize them.

1. ADARANK [9] is a listwise boosting approach aiming to optimize an arbitrary listwise IR metrics, such as the Mean Average Precision (MAP), ERR, or NDCG. Inspired by ADABOOST, it uses a stepwise greedy optimization technique for maximizing the chosen IR metrics. In every boosting iteration, ADARANK reweights the queries based on their scores obtained by the evaluation metrics: it up-weights the query having lower score and down-weights high-scoring queries. The weak learner is chosen by optimizing the listwise evaluation metrics of interest which is usually hard to optimize except for very simple weak classifiers. This can be viewed as a handicap of this method. According to the original implementation of ADARANK, we used the best feature ranker (BF) described above as base ranker taking into account the weighting of queries. The only hyperparameter of ADARANK is the number of boosting iterations which we optimized by using early-stopping on the validation set. We refer to this method as ADARANK.$\{$NDCG$\}$.
2. RANKNET [1] is a neural-net-based method which employs a loss based on pairwise cross entropy as its objective function. The neural net with one output node is trained to optimize directly the differentiable probabilistic pairwise loss instead of the common squared loss. We validated the number of hidden layers ranging from 1 to 3 and the number of neurons in the hidden layers ranging from 10 to 500. For the number of training epochs we applied early stopping.
3. RANKBOOST [4] is a pairwise boosting approach. The objective function is the rank loss (as opposed to ADABOOST which optimizes the exponential loss). In each boosting iteration the weak classifier is chosen by maximizing the weighted rank loss. For the weak learner we used decision stumps and a variant of the single decision stump described in [4] which is able to optimize the rank loss in an efficient way.
4. RANKSVM [5] is a pairwise method based on SVM, formulating the ranking task as a binary classification. We used linear kernel because the optimization using non-linear kernels cannot be carried out in reasonable time. The tolerance level of the optimization was set to 0.001 and the regularization parameter was validated in the interval $[10^{-6}, 10^4]$ with a logarithmically increasing step
5. COORDINATEASCENT (CA) [6] is a linear listwise model where the scores of the query-document pairs are calculated as weighted combinations of the feature values. The weights are tuned by using a coordinate ascent optimization method where the objective function is an arbitrary evaluation metrics given by the user. The coordinate ascent optimization method itself has two hyperparameters to be tuned: the number of restarts R from random initial weights, and the number of iterations T taken after each restart. We used $R = 30$ and $T = 100$. We did not validate these hyperparameters, but using the validation set we evaluated every model obtained due to restarting the optimization process, and we kept the one having highest performance.
6. LAMBDAMART [8] is a boosted regression tree model. Since it handles the LTR problem as a regression task, it could be classified as pointwise method, but during the training phase, it adjust the parameters of the regression trees based on the derivative estimate of NDCG, therefore it is considered as a listwise approach. We validated the number of boosting iterations. The number of leaves were set to 10 and the learning rate to 0.1.

5 Experiments

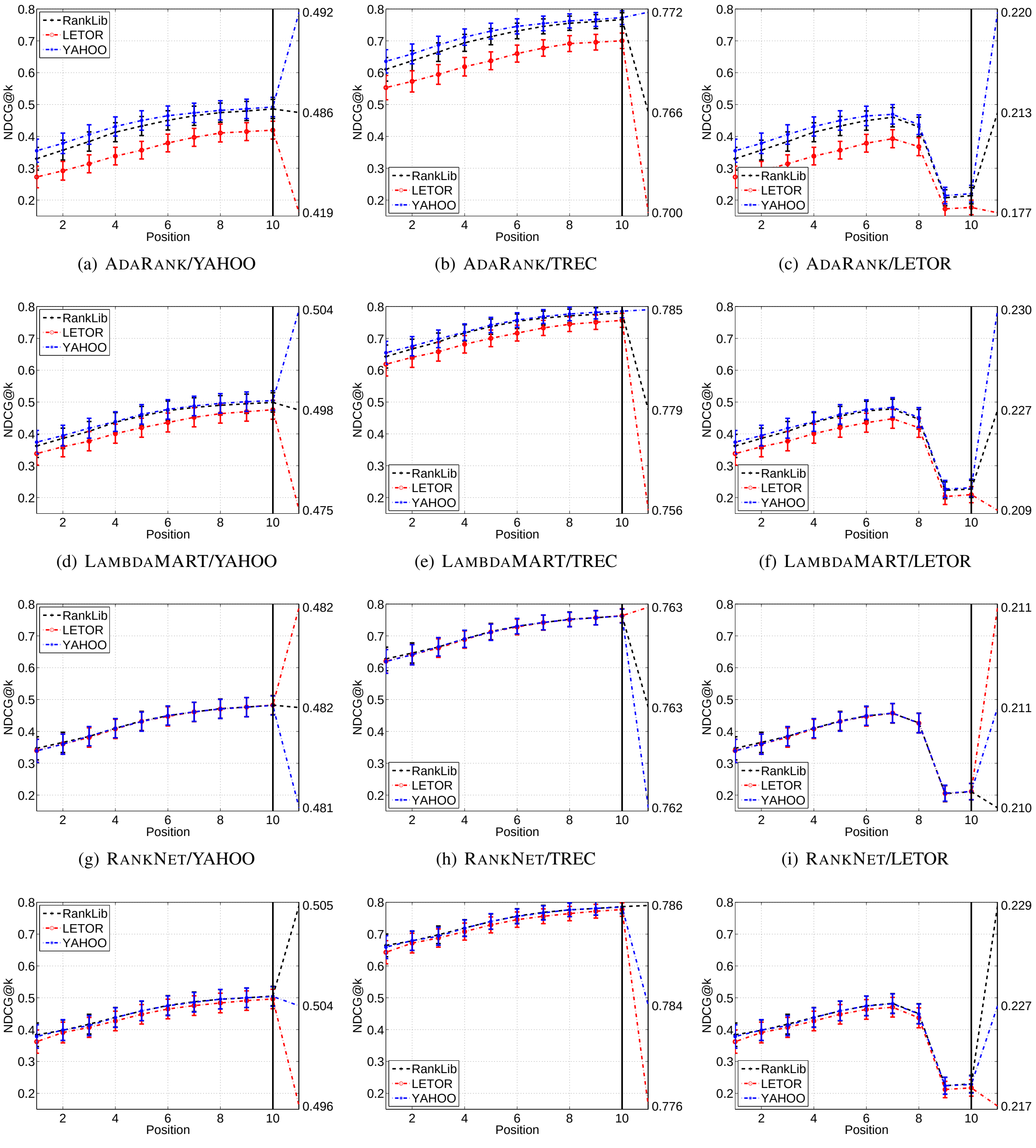
We identified two major differences in the way the DCG score is normalized in publicly available NDCG evaluation scripts:

1. When there is no relevant document for a query ($z_i = 0$ for all i), evaluation scripts conforming to the definition assign a 0.0 NDCG value to the query. Optionally, some scripts may assign a different value as default for such queries (namely, 1.0 for the Yahoo metric).
2. When the number of documents is less then the truncation level $n_{\max}$, evaluation scripts conforming to the definition assign an $\mathrm{NDCG}_{n^{(k)}}$ value equal to the number of documents $n^{(k)}$ available for that query (thereby assuming that it is always possible to fill in a result list with irrelevant documents, and at the same time assuming that the provided relevance labeling is exhaustive (i.e. there are no further, unseen relevant documents and thus the use of $\mathrm{IDCG}_{n^{(k)}}$ score in the normalization step is a realistic estimate.[14]) Optionally, some scripts may assign a different value as default for such queries (namely, 0.0 for the LETOR metric).

We consider the difference stemming from the different strategy to order results with equal predicted scores less crucial (in practice two different documents seldom get the same predicted score, i.e. ties are rare) and we do not assess its impact, and we use all metrics with exponential relevance grades (the TREC tool can be parameterized to use exponential gain, while the other tools are implemented to use this). That is, in our experiments we consider the Ranklib and TREC evaluation tools to be equivalent and compare their scores to those provided by the YAHOO and LETOR metrics, which are representative examples for the two major differences we found between the various tools.

In Figure 1, we plot the NDCG_{1-10} scores for 4 characteristic learning to rank algorithms (ADARANK, LAMBDAMART, RANKNET and CA) using the three different evaluation formulas to evaluate the models: in each row, the left, middle and right plots show the values provided by the RANKLIB/TREC, the YAHOO and the LETOR tools, respectively. All models were trained to optimize the NDCG_{10} scores, and the plots show the NDCG scores of these models for cut-off values from 1 to 10. Each plot shows 3 different curves that correspond to models trained using a specific metric during the training of the model: the blue, black and red lines indicate models learned using the YAHOO, RANKLIB/TREC and LETOR metrics, respectively. This way we can visually compare the effects

[14] Note that relevance labels are many times pooled in IR datasets and there is no guarantee that no relevant, but unlabeled documents exist.



(a) ADARANK/YAHOO

(b) ADARANK/TREC

(c) ADARANK/LETOR

(d) LAMBDAMART/YAHOO

(e) LAMBDAMART/TREC

(f) LAMBDAMART/LETOR

(g) RANKNET/YAHOO

(h) RANKNET/TREC

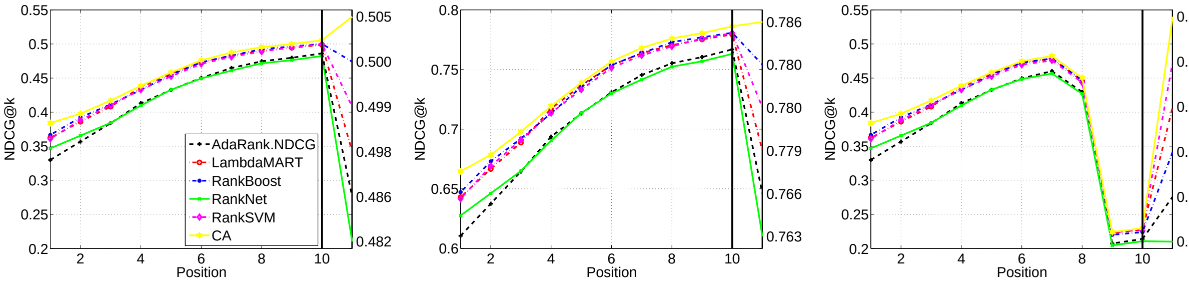
(i) RANKNET/LETOR

(j) CA/YAHOO

(k) CA/TREC

(l) CA/LETOR

Figure 1. The dependence of NDCG$_{10}$ scores on NDCG method used on MQ2008 dataset.



(a) YAHOO (b) TREC (c) LETOR

Figure 2. The dependence of NDCG$_{10}$ scores on NDCG method used on MQ2008 dataset.

of the different evaluation tools both on on the numeric outcomes (comparing the curves with same color, denoting the same models, across the 3 plots horizontally) and on the learnt model (contrasting the 3 curves on the same plots, denoting models learnt using different measures).

We can make several observations based on figure1. First, for all the algorithms in figure 1, the NDCG scores provided by the YAHOO tool are on average around 0.25 higher than the scores provided by the RANKLIB/TREC tool which conforms the definition of NDCG. This difference comes solely from the presence of queries with 0 relevant documents in the dataset: for those queries, the YAHOO tool assign 1.0 as NDCG which results in an additive bias compared to the official NDCG scores. On the other hand, the scores provided by the LETOR tool degrade rapidly for cutoff values of $8 - -10$ for all algorithms, with a difference around or above 0.25 compared to the official NDCG scores. As opposed to the former additive difference this behavior might result in a different ordering of the algorithms: a method that is on average superior to others on larger sets but is weaker than the competitors on small sets is preferred by the Letor metric, as this tool assigns zero to small sets (smaller than the NDCG cut-off).

Second, and more interestingly, we can observe that it is by no means irrelevant which metric was used during the training of the algorithms: for those algorithms that make use of the NDCG scores in some way in the training process (namely, the ADARANK, LAMBDAMART and CA methods), we see differences in the performance of the learnt models. We observe that in general, the LETOR tool is not suited for training the algorithms – its property to assign 0.0 score to small sets causes these small sets to be useless for training (to optimize NDCG$_{10}$), i.e. the algorithms can exploit less data in a meaningful way to learn patterns. This results in a significantly[15] worse performance for these algorithms, when trained using the LETOR metric (instead of the official NDCG scores). On the other hand, for some algorithms, it is worth to assign a non-zero score to sets which contain no relevant documents at all: doing so, the algorithms do not increase the weight of such queries similarly to other low-performing queries where there is hope to improve performance. This can result in better learning rates and in some cases slightly better performance (see e.g. the blue vs. black curves for the ADARANK and LAMBDAMART methods).

Overall, we can observe that the best metric to train the algorithms is the one provided by YAHOO: in general this results in equal or better learnt models than the other evaluation tools. To shed light on how these characteristics of the evaluation tools affect the relative performance of benchmark LTR algorithms, in figure 2, we plot the NDCG scores for 6 different learning algorithms (trained using the official NDCG metric) with all the evaluation tools surveyed in this study. As can be seen, the performance of the best algorithms is very close to each other, with Coordinate Ascent having a slight advantage regardless which metric is used for evaluation. Slightly worse than CA, the three algorithms RANKBOOST, RANKSVM and LAMBDAMART perform very close to each other. If we compare the results obtained with different evaluation measures, the relative order of these (otherwise very similar) models change depending on the metric: apparently the RANKBOOST algorithm has a slight advantage over the other two for short sets, which advantage disappears when the LETOR metric is used for evaluation. In general, we were unable to reproduce the competitive results of ADARANK [9], using the reimplementation of the algorithm in the Ranklib package [16],[17].

6 Conclusion

In this study, we reviewed the publicly available NDCG evaluation scripts, identified and compared the differences between them and systematically analyzed how these differences affect the numeric results and the training processes of various learning-to-rank algorithms. It is reasonable to assume that most previous studies use one of the assessed evaluation tools, and at the same time it seems likely that most measures are used at least in some studies to evaluate the performance of machine learnt ranking systems. We found that there indeed are differences between tools despite the relative simplicity of the popular NDCG evaluation metric, and our experiments demonstrate that these differences can easily lead to non-trivial differences between research results. Since most studies do not discuss such details as the evaluation script used, this fact makes previous studies very difficult to compare and might lead to the misinterpretation of results and false conclusions.

We identified that the two key points of difference between different tools are i) the way how small queries (queries with less than $n_{\max}$ documents in the case of NDCG$_{n_{\max}}$) and ii) the way how queries with no relevant document (i.e. when all documents have the same, zero relevance score and therefore the ideal DCG is zero) are handled by the tools. These two factors can lead to different overall scores (for the same model) and also to inherently different learnt models, depending on which learning algorithm was used. Some divergences from the definition of the NDCG measure might be jus-

[15] The error bars on the plots correspond to the standard errors of querywise NDCG scores averaged out in quadrature over the folds.

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

[17] We found no clear indication in the original article what stopping criterion was used to terminate the iteration in the training process, so we tried to use various stopping strategies and provide the best results we obtained. These are nevertheless lower than those reported at the letor website.

tified from an ML perspective though: for example, if the measure assigns a perfect score to queries with zero relevant documents, these queries are not treated like other queries that can be improved, and do not distort the model. On the other hand, we could not identify any benefit of other modifications, such as zeroing out the $\mathrm{NDCG}_{n_{\max}}$ score for queries with less than $n_{\max}$ documents in total.

To summarize our findings, we suggest the following protocols to make learning to rank results more comparable and benchmark datasets more suited to training and evaluating systems:

1. The use of tools that give zero score to small queries should be avoided, as these can negatively impact certain algorithms (while others are unaffected) and thus the reported results can represent a false relative order of the algorithms.
2. If possible, benchmark datasets should be free of not meaningful queries. Queries that does not have any relevant documents do not play a role in learning to rank – they cannot be used to learn meaningful patterns and they do not have an impact on evaluation. At the same time, such queries can negatively impact the performance of some algorithms, and can motivate non-standard evaluation tools (c.f. the YAHOO metric).
3. Regardless which evaluation script is used while training the systems (c.f. the YAHOO tool which is reasonable in the presence of queries without relevant documents in the data), the final evaluation should be carried out using a tool fully conforming the official NDCG definition. This way machine learnt performance scores would become directly comparable to non-machine learnt ones, such as those coming from TREC and other evaluation exercises.

REFERENCES

[1] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, ‘Learning to rank using gradient descent’, in *Proceedings of the 22th International Conference on Machine Learning*, (2005).

[2] R. Busa-Fekete, B. Kégl, T. Éltető, and Gy. Szarvas, ‘Tune and mix: learning to rank using ensembles of calibrated multi-class classifiers’, *Machine Learning* accepted, (2012).

[3] D. Cossock and T. Zhang, ‘Statistical analysis of Bayes optimal subset ranking’, *IEEE Transactions on Information Theory*, **54**(11), 5140–5154, (2008).

[4] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, ‘An efficient boosting algorithm for combining preferences’, *Journal of Machine Learning Research*, **4**, 933–969, (2003).

[5] R. Herbrich, T. Graepel, and K. Obermayer, ‘Large margin rank boundaries for ordinal regression’, in *Advances in Large Margin Classifiers*, eds., Smola, Bartlett, Schoelkopf, and Schuurmans, pp. 115–132. MIT Press, Cambridge, MA, (2000).

[6] D. Metzler and B. W. Croft, ‘Linear feature-based models for information retrieval’, *Information Retrieval*, **10**, 257–274, (2007).

[7] S. Robertson and H. Zaragoza, ‘The probabilistic relevance framework: BM25 and beyond’, *Found. Trends Inf. Retr.*, **3**, 333–389, (2009).

[8] Q. Wu, C.J.C. Burges, K.M. Svore, and J. Gao, ‘Adapting boosting for information retrieval measures’, *Information Retrieval*, **13**(3), 254–270, (2010).

[9] J. Xu and H. Li, ‘AdaRank: a boosting algorithm for information retrieval’, in *SIGIR ’07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 391–398. ACM, (2007).