Math. Appl. **7** (2018), 31–40
DOI: 10.13164/ma.2018.03

KERNELIZED COST-SENSITIVE LISTWISE RANKING

GABRIELA OLINTO AND ERNEST FOKOUÉ

Abstract. Learning to Rank is an area of application in machine learning, typically supervised, to build ranking models for Information Retrieval systems. The training data consists of lists of items with some partial order specified induced by an ordinal or binary score. The model purpose is to produce a permutation of the items in this list in a way which is close to the rankings in the training data. This technique has been successfully applied to ranking, and several approaches have been proposed since then, including the Listwise approach.

A cost-sensitive version of that is an adaptation of this framework which treats the documents within a list with different probabilities, i.e., attempt to impose weights for the documents with higher cost. We then take this algorithm to the next level by kernelizing the loss and exploring the optimization in different spaces.

Among the different existing likelihood algorithms, we choose ListMLE as primary focus of experimentation, since it has been shown to be the approach with the best empirical performance. The theoretical framework is given along with its mathematical properties.

1. Introduction

Ranking is a crucial problem in several different areas, especially in Information Retrieval (IR) and Machine Learning. Information Retrieval has its foundations on searching automated ways of reducing information overload. Web search engines are one of the most straightforward applications, but universities, government entities, and libraries use these systems to provide easier access to journals, documents, and books.

In a general scenario, the ranking problem can be defined as described by Lan et al. (2009): "Given a group of objects, a ranking model sorts the objects with respect to each other in the group, according to their degrees of relevance, importance, or preferences. Where in IR, a group corresponds to a query, and objects corresponds to documents associated with the query".

In other words, one could imagine a collection of queries with each query containing the n documents to be ranked. These documents could be ranked by a relevance score y where highly relevant documents are more useful and have higher ranks. Thus, one could formulate that the desire in ranking, is to construct a score function $f(x)$ such that the discrepancy between its ranking and the ordering of the relevance is as small as possible.

MSC (2010): primary 62J99, 62P30, 90C52.

Keywords: ranking, information retrieval, machine learning, statistics, loss function.

The author Gabriela Olinto acknowledges the financial and academic support given from CAPES and CNPq during this research.

Of course, "as small as possible" means the crucial need to define a suitable loss function, hopefully, one that is convex for the obvious optimization reasons. This indicates that this problem looks much like the tradition regression analysis, but for the traditional regression, the loss function is usually much more manageable and symmetric, whereas with ranking extra care is immediately required.

The form of $f(x)$ then becomes of great interest, just like with traditional regression, and various scenarios may be considered from linear to nonlinear to non parametric to kernelized. Because of that, the nature of the input space can quickly become a problem because the data is usually initially unstructured and is then processed in ways that could lead to mathematical challenges. We will later consider a common practice [7] of further preprocessing $|f(x)| < BM$ where BM are normalization constants.

These models can be categorized into three main groups: pointwise, pairwise and listwise. Pointwise and pairwise directly transform ranking into ordinal regression, and classification on a single object or object pairs respectively [2]; and listwise that minimizes the loss function based on the ranked list and the ground truth list [16].

This paper focuses on the cost-sensitive listwise approach [10] that relies on changing the weighting scheme of the document pairs which are measured by their ranking effectiveness with the Normalized Discounted Cumulative Gain (NDCG) [3]. Based on that we propose a theoretical improvement by including a Gramian matrix inside the loss function. The kernel trick allows the implicit computation of feature space calculations with functions (kernels) defined in the input space which will enable high-dimensional operations and consequently greater flexibility.

The early methods of optimization (minimization) of nonlinear functions were developed from the basic idea of making the algorithm evolve by finding new points located in the direction to which the function decreases compared to the current point. These optimization methods are compared to each other according to the number of evaluations of the objective function which are required for determining the solution and according to how close the solution provided by this method approaches to the exact solution of the problem. The best method is found by the fewer necessary assessments and close it gets. Each new point is obtained from one-dimensional optimization process having as a start the previous one.

The first reasonable choice for a search direction in the k-th step, d_k, is the opposite direction to the gradient function in the current point x_k. This choice is justified by the fact that, locally, this is the direction in which the function decreases faster. The only implicit assumption in the application of this algorithm is that the function $f(x)$ is twice differentiable. Notice that the Hessian indicates the step size to achieve a reduction in f while still making sufficiently fast progress towards the optimal solution. Thus, if the Hessian is a well-behaved matrix, i.e., is positive definite, it can give optimal step sizes towards the right direction and make the convergence even faster. On the other hand if it is an ill-conditioned matrix it can do a zig-zag trajectory and take longer to converge to the minimum. In some cases, when the Hessian cannot be calculated, its value is substituted by the learning rate – a parameter that determines how fast the optimal solution is approached; this makes such cases treatable, albeit challenging. With all that said,

Gradient Descent is a perfect tool for optimization Empirical Risk Functions and will be chosen for optimizing our losses for ranking problems.

Some experiments are conducted to validate this new framework using the dataset MSLR-WEB10K in Microsoft Learning to Rank Datasets inside Letor [9]. This new framework is called Kernel Cost-Sensitive Listwise MLE (Kernel CS-ListMLE) and we validate it by comparing to its simpler version Cost-Sensitive Listwise MLE (CS-ListMLE) [10] and to other two well known techniques in the literature ListNet [16] and RankSVM [5]. We also explore different aspects of the kernels inside the Kernel CS-ListMLE. All the variants of the CS-ListMLE (plain/kernel) were implemented using the R project [15], the ListNET can be found on RankLib [1] on the Lemur project, and RankSVM can be found on SVM^{rank} [4]. RankLib and RankSVM are free standalone libraries implemented in Java and Oracle respectively.

This paper is organized as follows. Section 2 illustrates Kernelized Listwise Ranking framework, an evaluation metric and details on the implementation. Section 3 shows data details, experiments, and comparison results. Section 4 outlines future points that could be extended from this paper and draws conclusions.

2. Kernelized Listwise Ranking

The process in Information Retrieval [11] starts when a user enters a query into the system. This query matches several documents in a collection, maybe with different degrees of relevancy, and these results are ranked. The query can be for example a search string in web search engine and the documents, for example, text documents, images, audio or videos. Most of this systems compute a numeric score on how well each object in the database matches an individual query and rank the objects according to this value. The top ranking objects are then shown to the user.

There are many of standard metrics to judge how well the learn to rank models are doing on training the data and also to compare how they perform between algorithms [3]. These metrics take into account the relative order of the documents retrieved by the system, and give more weight to documents returned at higher ranks. Often these models are reformulated as an optimization problem on one of these metrics. The most used metric is the Normalized Discounted Cumulative Gain (NDCG) [3].

Considering rel_i as the relevance of the result at position i, DCG or Discounted Cumulative Gain measures the gain of a document based on its position in the result list of documents retrieved from a query

$$DCG = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i+1)}.$$

Because the length of the query results usually differs from one query to the other, we normalize them by dividing it by the maximum possible DCG on that query so the measure can be comparable between lists. What this means is that we actually need to recalculate the above metric but this time with best ranked

ordered on that list and get:

$$NDCG = \frac{DCG}{\max(DCG)}.$$

We will also fix the number of returned results since this version does not penalize for missing results, thus DCG will become $DCG@k$. There are different classes of ranking algorithms in the literature but we will focus on Listwise and here is how we can define this approach. Consider the query-level framework proposed by [8]:

Let Q, D and X be respectively the query space, the document space and the d-dimensional feature space. Let Π be a random variable defined in the query space with unknown probability distribution P_Q. Let f denote the real value ranking function, which assigns each document a score $f(x)$. The scores of the documents associated with the same query are used to rank the documents. We measure the loss of ranking documents for query q using f with a loss function $L(f; q)$. The goal of ranking is to minimize the expected risk of f.

Represent query q by (z, y), where $z = (x_1, \ldots, x_m)$ and y stands for the ground-truth permutation of m documents. Let $Z = X^m$ be the input space, whose elements are m feature vectors corresponding to the m documents, where $y(i)$ stands for the index of the document whose rank is i in the permutation y. We call m the list length and assume that $m \geq 3$. Let $\mathcal{Y}$ be the output space, whose elements are permutations of the m documents. Then we regard (z, y) as a random variable on the space $Z \times Y$ according to an unknown probability distribution $P(\cdot, \cdot)$.

The goal of ranking is to minimize the expected risk of f. Thus, rewriting $L(f; q) = l(f; z, y)$ and $P_Q = P(\cdot, \cdot)$, it can be defined as:

$$R_t(f) = \int_{Z \times Y} l(f; z, y) P(dz, dy).$$

Since $P(\cdot, \cdot)$ is unknown, the empirical risk is used to approximate the expected risk, and is defined as:

$$\hat{R}_t(f; S) = \frac{1}{n} \sum_{i=1}^{n} l(f; z, y)$$

where i denote the i.i.d sampled training data from the space $Z \times Y$, $S = \{(z_1, y_1), \ldots, (z_n, y_n)\}$ and $z_i = (x_1^{(i)}, \ldots, x_m^{(i)})$.

One of the most critical analyses is the generalization ability is to find a tight upper bound of $\sup_{f \in \mathcal{F}}(R_l(f) - \hat{R}_l(f; S))$ which for this case can be obtained by applying the theory of Rademacher Average [12] that measures how much a function class $\mathcal{F}$ can fit random noise and it is showed by the following theorem proved in [6].

Theorem 2.1. *Let Λ denote a listwise ranking algorithm, and let $l_\Lambda(f; z, y) \in [0, 1]$ be its listwise loss, given the training data $S = \{(z_i, y_i), i = 1, \ldots, n\}$, with probability at least $1 - \delta$, the following inequality holds:*

$$\sup_{f \in \mathcal{F}}(R_l(f) - \hat{R}_l(f; S)) \leq \sqrt{\frac{2 \ln(2/\delta)}{n}} + 2\hat{\mathcal{R}}(l_\Lambda \circ \mathcal{F})$$

where $\hat{\mathcal{R}}(l_\Lambda \circ \mathcal{F}) = E_\sigma \ \sup_{(f\in\mathcal{F})} \ \frac{1}{n}\sum_{i=1}^{n} \sigma_i l_\Lambda(f; z_i, y_i)$.

There are many Listwise loss functions widely studied but we will focus on ListMLE because it holds theoretical properties given by its surrogate loss, including consistency, soundness, continuity, differentiability, convexity and efficiency which we will not prove in this paper for lack of space.

$$\textbf{ListMLE} : l(f; z, y) = -\log P(y \mid z; f)$$
$$P(y \mid z; f) = \prod_{i=1}^{m} \frac{\phi(f(x_{y(i)}))}{\sum_{j=1}^{m} \phi(f(x_{y(j)}))}.$$

Two assumptions are further made on the feature vector and the ranking model for implementation purposes. Let x be the feature vector of a query document-pair, we assume that $\forall x \in X$, $||x|| \leq M$ and the ranking model f to be learned is from the linear function class $F = \{x \to w \cdot x : ||w|| \leq B\}$. Therefore, we have $\forall x \in X$, $\forall f \in F$, $|f(x)| \leq BM$. With this normalization, the algorithms can minimize the empirical risk in learning. Note that this normalization does not affect their optimal solution.

Although Listwise MLE appears to work very well, its use of a linear representation for the score function could be a limitation. Kernels can help capture an arbitrarily nonlinear function by mapping into a higher dimension space. One way to do it so is by a feature mapping.

The kernel trick allows us to do that in a more efficient way; instead of a transformation for each predictor, we apply a single similarity function over the data and enable them to operate in a higher dimensional space without ever computing a transformation for each of the coordinates. Instead we can simply compute the inner product between the images of all pairs of data in the feature space.

Mathematically, let the vector $x \in \mathbb{R}^p$ implicitly be projected (mapped) onto a feature space F (of potentially infinite dimension) via a transforming function or mapping η such that $x \to \eta(x)$, and $K(x_i, x_j) = \eta(x_i)^T \eta(x_j)$. Many common choices of kernels work very well in practice, some frequently used ones are:

- **Polynomial**: $K(x_i, x_j) = (bx_i^T x_j + a)^d$;
- **Gaussian**: $K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$;
- **Laplace**: $K(x_i, x_j) = \exp\left(-\gamma\|x_i - x_j\|^2\right)$;
- **Hyperbolic Tangent**: $K(x_i, x_j) = \tanh(bx_i^T x_j + a)$.

With all that said it is easy to extend the Cost-Sensitive Ranking [10] for a Kernel Cost-Sensitive Ranking by kernelizing the ranking function $f(x)$, i.e., let

$$\begin{cases} \eta : & X \to F \\ & x \to \eta(x) \end{cases}$$

so $f(x) = \theta^T \eta(x)$ in F. Then $f(x) = \sum_{j=1}^{n} \theta_j K(x, x_j)$ where $K(\cdot, \cdot)$ is the kernel trick which make it inherit all its good characteristics. The loss function on a query

can be defined as:

$$L = \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \beta_j \times$$
$$\times \log_2 \left(1 + \sum_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp\left(\sum_{l=1}^{n} \theta_l K(x_t, x_l) - \sum_{l=1}^{n} \theta_l K(x_j, x_l)\right) \right).$$

The Gradient Descent method can be used to optimize the loss function which is a method that works optimally when is based on the gradient and the Hessian. It is vital to assess the properties of the Hessian matrix because its positive definiteness helps estimating the vector θ.

Thus, let $G(\theta)$ be the gradient; then, with our loss function L, we can write $G(\theta) = (G(\theta_1), \ldots, G(\theta_p))$ as:

$$G(\theta_l) = \frac{\partial L}{\partial \theta_l} = \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \frac{\beta_j}{\log(2)} \times$$
$$\times \frac{\sum_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[f(x_t) - f(x_j)] \sum_{l=1}^{n} [K(x_t, x_l) - K(x_j, x_l)]}{\left(1 + \sum_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[f(x_t) - f(x_j)]\right)}$$

and let $H(\theta)$ be the hessian then with the loss function L, we can write $H(\theta) = (H(\theta_1), \ldots, H(\theta_p))$ as:

$$H(\theta_l) = \frac{\partial^2 L}{\partial \theta_l^2} = \frac{1}{DCG_{\hat{g}}@k} \sum_{j=1}^{n} \frac{\beta_j}{\log(2)} \times$$
$$\times \frac{\sum_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[f(x_t) - f(x_j)] \left(\sum_{l=1}^{n} [K(x_t, x_l) - K(x_j, x_l)]\right)^2}{\left(1 + \sum_{t=j+1, y_j > y_t}^{n} \alpha_{j,t} \exp[f(x_t) - f(x_j)]\right)^2}$$

considering

$$f(\cdot) = \sum_{l=1}^{n} \theta_l \eta(\cdot) \eta^T(x_l) = \sum_{l=1}^{n} \theta_l K(\cdot, x_l).$$

The algorithm for Gradient Descent in Kernelized CS-ListMLE is shown in Algorithm 1.

For the kernels used in this case in which the results are shown in Section 3, the Hessian appears to behave very well, making the convergence faster, especially in concert with the normalization of the input space. Since this version is based on the same surrogate loss the theoretical properties of the listwise carry over to the kernelized version.

Algorithm 1 GD for Kernelized CS-ListMLE

1: Input normalized training set, tolerance rate ϵ, NDCG@k
2: Initialize θ and compute weights with respect to each query, normalizing θ each step
3: Do Gradient Descent until the change of the loss function is less than ϵ
4: Return θ

It is important to point out that this nonlinear approach benefits from higher dimensional projections which makes an easy task to fit a broad range of functions since it can search the space better. Although many scientific processes can be described well using linear models, others are inherently nonlinear. Another benefit is efficient use of data. Nonlinear regression like models can produce good estimates of the unknown parameters in the model with relatively small data sets.

The major cost of moving to nonlinear from simpler modeling techniques is the need to use iterative optimization procedures to compute the parameter estimates. The use of iterative procedures requires the user to provide starting values for the unknown parameters before the software can begin the optimization and it is directly tied up to its convergence. Bad starting values can also cause the software to converge to a local minimum rather than the global minimum that defines the least squares estimates.

In our case, there is no analytic solution for the Listwise approaches and the convexity of the loss working in tandem with normalization of the space make this approach very appealing.

3. Data, experiments and results

To prove our results we use one of the datasets in the LETOR project. The LETOR dataset is a benchmark collection for research on learning to rank for information retrieval, released by Microsoft Research Asia [14]. By doing this, the group facilitates the research on learning to rank since no selected queries for training and test, no standard feature vectors, no baseline algorithms and no standard evaluation tools were defined before, making the comparison among different methods not possible. Their most recent release was the $MSLR-WEB10K$ with 10000 queries and 136 features for each query-url pair which are available to download [13]. Taking a closer look in the data files, each row corresponds to a query-url pair. The first column is relevance label of the pair which goes from 0 (irrelevant) to 4 (perfectly relevant), the second column is query id, and the following columns are features.

In this section we analyze the performance of ListNet, RankSVM, Plain Cost-Sensitive MLE and Cost-Sensitive MLE Kernel polynomial, gaussian, laplace and hyperbolical tangent on the LETOR dataset $MSLR-WEB10K$.

We also explore some different parametrizarion for the kernels, one of them we call **special** in which the offset is considered 0, the polynomial degree is 3 and the bandwidth is $1/p = 1/136 = 0.0074$, p being the number of features in the data. In the other cases we use the vanilla parametrization with offset is considered 0, the polynomial degree is 1 and the bandwidth is 1.

It is important to notice when generating test results that ListNet and Plain Cost-Sensitive MLE optimize distances while RankSVM and Cost-Sensitive MLE Kernel optimize similarities which means different interpretations for the coefficients. In the experiments, the truncation level k takes 3, 5 and 10 as in [10].

Table 1. Test results for (a)nDCG@3, nDCG@5 and nDCG@10 on ListNet and RankSVM (b) nDCG@3 on Kernel and Plain Cost Sensitive ListMLE (c) nDCG@5 on Kernel and Plain Cost Sensitive ListMLE (d) nDCG@10 on Kernel and Plain Cost Sensitive ListMLE.

(a)

nDCG	ListNet	RankSVM
@3	**0.6028**	0.4662
@5	**0.6243**	0.5120
@10	**0.6116**	0.5325

(b)

Methods	at.3	at.5	at.10	special
Gaussian	0.5763	0.5736	0.5693	0.5548
Laplace	0.5765	0.5673	0.5705	0.5637
Polynomial	**0.6011**	**0.6040**	0.6178	0.5126
Tanh	0.5824	0.5904	0.5666	**0.5743**
Plain	0.5817	0.5984	**0.6316**	

(c)

Methods	at.3	at.5	at.10	special
Gaussian		0.5948	0.5933	0.5755
Laplace		0.5882	0.5984	0.5850
Polynomial		**0.6177**	0.6293	0.5455
Tanh		0.6058	0.5947	**0.5986**
Plain		0.6122	**0.6425**	

(d)

Methods	at.3	at.5	at.10	special
Gaussian			0.5860	0.5757
Laplace			0.5876	0.5813
Polynomial			0.6175	0.5579
Tanh			0.5941	**0.5917**
Plain			**0.6276**	

Note that the empty values on the tables are because the evaluation metric can just evaluate cases where its truncation value is equal or bigger than training truncation level. At the special column, no parameter tuning can be done for the Plain version, so it is always empty.

Since ListNet and RankSVM do not truncate during training we should compare our results with highest truncation at training (at 10) and same nDCG metric to have somewhat fair comparisons. With that said one could notice that CS-ListMLE plain and CS-ListMLE polynomial always perform better than them which reinforce the point of our study.

Looking at the test results for nDCG@3, we can see for a truncation level during training also at 3, that the cost-sensitive listwise Polynomial outperforms the others. Considering higher truncation levels but same metric, Polynomial, Plain and tanh outperforms the others at 5, 10 and 10 special respectively.

For test results for nDCG@5 with the same truncation level 5, the Polynomial kernel again outperform the others. Considering higher truncation at training as 10 and special, Plain and Tanh outperforms the others respectively.

Finally, the test results for nDCG@10 shows that the Plain version outperforms with truncation level at 10 although Polynomial kernel also does a very good job and for the special case, Tanh as usual does the best job.

Taking a look at how many iteration it takes to converge, we can say that it is an algorithm that converges relatively fast which is probably due to the fact that the loss function is convex and the Gradient Descent include the second derivative. Considering the truncation level 3, Laplace and Plain are the fastest,

at level 5 Gaussian and Plain, at 10 Laplace and Tanh and at 10 special we have got Polynomial.

These results lead us to conclude that CS-MLE Plain is more robust when evaluating larger truncation levels than it was trained on but the CS-MLE Kernel outperforms cases in which the truncation and the evaluation are the same, that means, it learns better its own space.

More parameters could have been tuned, but our main goal was to show the potential behind the Kernelized options, which can be seen from the standard version. Specially for the Tanh kernel, its tests results seem to gain the most.

4. Conclusion and future work

This paper studied new approaches for cost-sensitive listwise to learning to rank which is an area of application in machine learning that built ranking models for Information Retrieval systems. The model purpose is to produce a permutation of items that have some partial order induced by an ordinal score. The listwise approach learns a ranking function by taking lists and minimizing a loss function based on the ranked list and the ground truth list (supervised annotated judgment). These losses are classified in different groups with different inherent properties: we verify that the surrogate likelihood loss is the best one and study the particular ListMLE case.

A cost-sensitive version is the next straightforward step, using the traditional nDCG@k metric, which we generalize by introducing the Gram matrix to the loss or kernelizing it. The theoretical framework was given along with their mathematical properties. Four families of kernels were considered: Gaussian, Laplace, Polynomial and Hyperbolic Tangent. We use their standard versions to compare with the baseline plain method as well as some different parameterizations commonly utilized for the kernel. The carry parameters were set as offset 0, polynomial degree 3 and bandwidth $1/p = 0.0074$.

The experimentation is done on the benchmark LETOR $MSLR - WEB10K$ dataset. It is the most recent available dataset released by the Microsoft group. They contain queries and some characteristics of the retrieved documents and its human judgments on the relevance of the documents with respect to the queries and its natural five fold set allowed cross validation.

Gathering all those ideas we have shown some higher performance Kernel Cost-Sensitive ListMLE compared to the baseline ListMLE and compared different aspects of the proposed loss function within different families of kernels. The results show that CS-MLE Plain is more robust when evaluating larger truncation levels than it was trained on, but the CS-MLE Kernel outperforms cases in which the truncation and the evaluation the same, i.e., it better learns its space.

More grid values for parameters could have been studied; however, our goal was to show the potential behind the Kernelized options, and the gain from this method could be seen in its standard version. Especially for the Tanh kernel, its tests results seem to gain the most.

There is clear many future directions to enrich this area of research. One direct extension of this research is to change the way the ensemble of the losses is done in the Gradient Descent training. Another area of the investigation is to modify

the loss itself to make the kernels learn the whole space of the queries instead of looking it query by query. Finally, one could analyze how the theoretical bounds change with CS-MLE Kernel.

References

[1] V. Dang, *The Lemur Project / Wiki / RankLib*, Lemur Project, `http://sourceforge.net/p/lemur/wiki/RankLib`.

[2] R. Herbrich, T. Graepel and K. Obermayer, *Large margin rank boundaries for ordinal regression*, in: Advances in Neural Information Processing Systems, MIT, 2000, 115–132.

[3] K. Järvelin and J. Kekäläinen, *Cumulated gain-based evaluation of IR techniques*, ACM Transactions on Information Systems **20** (2002), 422–446.

[4] T. Joachims, *SVM^{rank} Support Vector Machine for Ranking*, SVM^{rank}, `https://www.cs.cornell.edu/people/tj/svm_light/svm_rank.html`.

[5] T. Joachims, *Training linear SVMs in linear time*, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2006, ACM, 217–226.

[6] Y. Lan, T.-Y. Liu, Z. Ma and H. Li, *Generalization analysis of listwise learning-to-rank algorithms using Rademacher average*, CiteSeer^{x}, 2008.

[7] Y. Lan, T.-Y. Liu, Z. Ma and H. Li, *Generalization analysis of listwise learning-to-rank algorithms*, in: Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, ACM, 2009, 577–584.

[8] Y. Lan, T.-Y. Liu, T. Qin, Z. Ma and H. Li, *Query-level stability and generalization in learning to rank*, in: Proceedings of the 25th International Conference on Machine Learning, ACM, 2008, 512–519.

[9] T.-Y. Liu, J. Xu, T. Qin, W. Xiong and H. Li, *Letor: Benchmark dataset for research on learning to rank for information retrieval*, in: Proceedings of SIGIR 2007 workshop on learning to rank for information retrieval, 2007, 3–10.

[10] M. Lu, M. Xie, Y. Wang, J. Liu and Y. Huang, *Cost-sensitive listwise ranking approach*, in: M. J. Zaki, J. X. Yu, B. Ravindran and V. Pudi (eds.), Advances in Knowledge Discovery and Data Mining, Part I, 14th Pacific–Asia Conference, PAKDD 2010, Hyderabat, India, June 21–24, 2010, Proceedings, Lecture Notes in Artificial Intelligence **6118**, Springer-Verlag, Berlin Heidelberg, 2010, 358–366.

[11] C. D. Manning, P. Raghavan, H. Schütze, *Introduction to Information Retrieval*, Cambridge University Press, Cambridge, 2008.

[12] S. Mendelson, *Rademacher averages and phase transitions in Glivenko–Cantelli classes*, Information Theory, IEEE Transactions on Information Theory **48** (2002), 251–263.

[13] T. Qin, T.-Y. Liu, W. Ding, J. Xu and H. Li, *Microsoft Learning to Rank Datasets*, `http://research.microsoft.com/en-us/projects/mslr/`.

[14] T. Qin, T.-Y. Liu, J. Xu and H. Li, *LETOR: A benchmark collection for research on learning to rank for information retrieval*, Information Retrieval **13** (2010), 346–374.

[15] R Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2015.

[16] F. Xia, T.-Y. Liu, J. Wang, W. Zhang and H. Li, *Listwise approach to learning to rank: Theory and algorithm*, in: Proceedings of the 25th International Conference on Machine Learning, ICML '08, ACM, 2008, 1192–1199.

Gabriela Olinto, School of Mathematical Sciences, College of Science, Rochester Institute of Technology, 1 Lomb Memorial Dr, Rochester, NY 14623, United States
e-mail: `ggo5219@rit.edu`

Ernest Fokoué, School of Mathematical Sciences, College of Science, Rochester Institute of Technology, 1 Lomb Memorial Dr, Rochester, NY 14623, United States
e-mail: `epfeqa@rit.edu`