On Including the User Dynamic in Learning to Rank

Nicola Ferro
University of Padua, Padua, Italy
ferro@dei.unipd.it

Claudio Lucchese
ISTI–CNR, Pisa, Italy
claudio.lucchese@isti.cnr.it

Maria Maistro
University of Padua, Padua, Italy
maistro@dei.unipd.it

Raffaele Perego
ISTI–CNR, Pisa, Italy
raffaele.perego@isti.cnr.it

ABSTRACT

Ranking query results effectively by considering user past behaviour and preferences is a primary concern for IR researchers both in academia and industry. In this context, LtR is widely believed to be the most effective solution to design ranking models that account for user-interaction features that have proved to remarkably impact on IR effectiveness. In this paper, we explore the possibility of integrating the user dynamic directly into the LtR algorithms. Specifically, we model with Markov chains the behaviour of users in scanning a ranked result list and we modify LambdaMart, a state-of-the-art LtR algorithm, to exploit a new discount loss function calibrated on the proposed Markovian model of user dynamic. We evaluate the performance of the proposed approach on publicly available LtR datasets, finding that the improvements measured over the standard algorithm are statistically significant.

CCS CONCEPTS

•Information systems →Learning to rank; Query log analysis; Retrieval effectiveness;

KEYWORDS

LambdaMart; learning to rank; user dynamic

ACM Reference format:
Nicola Ferro, Claudio Lucchese, Maria Maistro, and Raffaele Perego. 2017. On Including the User Dynamic in Learning to Rank. In *Proceedings of SIGIR '17, Shinjuku, Tokyo, Japan, August 07-11, 2017,* 4 pages.
DOI: http://dx.doi.org/10.1145/3077136.3080714

1 INTRODUCTION

Information Retrieval (IR) systems are nowadays challenged with increasingly complex search tasks where information about how users interact with IR systems play a central role to adapt them to user needs and interests [13]. A lot of IR research focused on improving effectiveness, by exploiting information about user-system interactions recorded in the query logs of Web search engines. The number of clicks on a given query-result pair, the click-through rate, and the dwell time, are examples of actionable information to improve various aspects of IR systems. In the context of *Learning to Rank (LtR)*, user actions recorded in query logs are used to extract several important features [1]. As an empirical evidence of the importance of user interaction features, we trained a LambdaMart [4, 19] model on the MSLR-WEB10K LtR dataset (https://www.microsoft.com/en-us/research/project/mslr/) with and without user-interaction features: the nDCG measured on the test set without such features drops from 0.4636 to 0.4410.

In this paper, we explore the embedding of user interaction dynamics into LambdaMart, a state-of-the-art LtR algorithm. Instead of proposing new features modeling this particular aspect of user behavior and then training the LtR model on this extended set of features, we adopt a complementary approach. We model the user dynamic in scanning a ranked result list with Markov chains trained on query log data and we modify the LambdaMart loss function to embed this trained Markov chain.

To the best of our knowledge, the integration of the user dynamic in a LtR algorithm is novel and has not been addressed yet. Our approach differs in fact from on-line LtR and reinforcement learning. The authors of [11] exploit click logs to infer preferences between rankers in order to make on-line LtR faster. Lerot [16] proposes an on-line LtR algorithm which uses clicks as feedback for interleaving methods. We instead propose an off-line LtR algorithm where the user dynamic is directly embedded in the ranking function.

2 METHODOLOGY

As shown in [20], effectiveness is often measured as the inner product of a relevance vector $\mathcal{J}$ and a discounting vector $\mathcal{D}$. The elements $\mathcal{J}_i$ account for the benefit of ranking an high-quality document at the i-th position of the *Search Engine Result Page (SERP)*, while $\mathcal{D}$ denotes such contribution for low-ranked documents. For instance, according to *Discounted Cumulated Gain (DCG)* the i-th element of $\mathcal{J}$ is defined as $\mathcal{J}_i = 2^{l_i} - 1$, where l_i is the relevance label of the i-th ranked document, and $\mathcal{D}_i = \log(i+1)$. The underlying assumption is that low-ranked documents receive less attention by the user and therefore they contribute less to the user-perceived quality of the SERP. Defining a proper quality metric is crucial both for evaluating retrieval systems and for learning effective ranking models as such metrics are used to drive the training process.

Most metrics assume the user analyzes a SERP from top to bottom, and therefore define a decreasing discount vector, hoever some user studies suggest that the probability of observing a result depends on the quality of the documents ranked higher: if the user finds a relevant document at position i it is less likely that he will

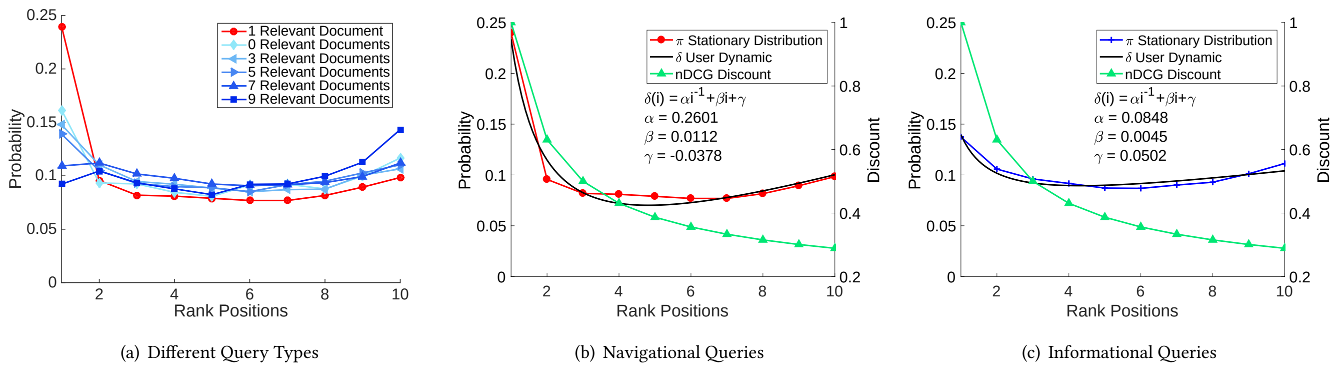

Figure 1: Stationary distributions for queries retrieving different number of relevant documents (a), and stationary distribution with its fitted curve and DCG discount for navigational (b), and informational queries (c).

inspects the document at position $i + 1$ [21]. However, the user behavior is more complex as he/she can move forward and backward, can jump from one document to any other and visit already visited documents, as suggested by [10].

Our work stems from the simple observation that user behavior in visiting a SERP differs depending on query type and the number of relevant results. For example, it is likely that on a SERP with a single highly relevant result in the first position the user assumes a *navigational* behavior, while a SERP with several relevant results may likely correspond to an *informational* query, where a more complex SERP visiting behavior can be observed [2]. Since at training time a list-wise LtR algorithm such as LambdaMart is aware of the number and distribution of relevance labels associated with the training samples for each query, we suppose that it can profit from the knowledge of the user dynamic associated with the specific kind of query. In the following we discuss our model of user dynamic and the methodology followed to integrate it into LambdaMart.

2.1 Modeling User Dynamic

We model the user dynamic with a Markovian process [14] where the user scans the ranked documents in the SERP according to possibly complex paths. Let us denote by $X_1, X_2, \dots$ the sequence of random variables representing the rank positions in $\mathcal{R} = \{1, 2, \dots, R\}$ visited by the user, where $X_n = j$ means that the n^{th} document visited by the user is at rank j. Moreover, we assume that the probability to move from the document at rank i to the document at rank j depends on the document at rank i only and is independent of all the previously visited documents. Finally, we denote by P the transition matrix whose entries represent the transition probabilities $P = (p_{ij} : i, j \in \mathcal{R})$, where $p_{ij} = \mathbb{P}[X_{n+1} = j | X_n = i]$. The sequence of random variables $(X_n)_{n>0}$ defines a discrete-time homogeneous Markov chain. Under the assumption of irreducibility and aperiodicity, P admits a unique stationary distribution $\pi = \pi P$, which is the limit of the n-step transition probabilities $p_{ij}^{(n)} \to \pi_j$ as $n \to \infty$ for all i, j [14]. When extending this analysis to a long-term query log, we can consider the behavior recorded for each user as a different observation of the same stochastic process, and the resulting stationary distribution can be considered as an aggregated representation of user dynamics. In addition, since we observe that the behavior of users change depending on the number of relevant documents in the SERP, we can classify queries on the basis of the number of relevant documents returned and estimate different transition matrices $\hat{P}$ for different classes of queries. Specifically, we first aggregate the dynamics of different users on the basis of the typology of query, then we adopt the maximum likelihood estimator approach [18] on the aggregated data:

(1) for each $i \in \mathcal{R}$ let v_i be the number of times that the users visited the document at rank i given the query;
(2) if $v_i = 0$, then $\hat{p}_{ij} = 0$ for all $j \neq i$ and $\hat{p}_{ii} = 1$;
(3) if $v_i > 0$, let v_{ij} be the number of transitions from document at rank i to document at rank j, then $\hat{p}_{ij} = \frac{v_{ij}}{v_i}$.

Figure 1(a) plots the stationary distributions obtained from the Yandex query log detailed in Section 3.1. When considering queries with just one relevant retrieved document, i.e. the red line with circle markers in Figure 1(a), the user dynamic exhibits a spike with respect to the first rank position, while for queries without any relevant documents or with more than one relevant document, i.e. the blue lines, the probability tends to be distributed more uniformly, meaning that the user is exploring the whole SERP.

We focus on these two distinct macroscopic behaviors, and, for the sake of simplicity, we call *navigational* the queries where users concentrated on just the first item, and we consider all the other queries as *informational* since users tend to visit more documents.

On the basis of the above experimental observation, we claim that the user dynamic can be described as a mixture of the navigational and informational behavior. The navigational component is represented by the inverse of the rank position i, $\frac{1}{i}$, while the informational component is linear with respect to the rank position i. Therefore, we model the *user dynamic* as $\delta(i) = \alpha i^{-1} + \beta i + \gamma$, where the parameters α, β and γ are calibrated in order to fit the estimated stationary distributions computed on the Yandex dataset.

Figures 1(b) and 1(c) show the stationary distributions together with the fitted curves for the navigational and informational cases, respectively. In Figure 1(b) the stationary distribution is the same reported in the red line of Figure 1(a), while to compute the stationary distribution reported in Figure 1(c) we aggregate all the user dynamics corresponding to the other queries, i.e. queries without relevant documents or with more than one relevant document.

The *user dynamic* defined above can actually be considered as a discounting vector to be exploited in any given quality metric. Differently from other approaches, the *user dynamic* is defined on the basis of two different query classes which exhibit a different user behavior. Figures 1(b) and 1(c) show how different is the derived user dynamic w.r.t. the DCG discounting component. Below we discuss how δ can be exploited in a state-of-the-art LtR algorithm.

2.2 Integrating User Dynamic into LtR

A LtR algorithm exploits a *ground-truth* set of training examples in order to learn a document scoring function σ [12]. Such training set is composed of a collection of queries Q, where each query $q \in Q$ is associated with a set of assessed documents $D = \{d_0, d_1, \ldots\}$. Each document d_i is labeled by a *relevance judgment* l_i according to its relevance to the query q. Each query-document pair (q, d_i) is represented by a vector of features x, able to describe the query (e.g., its length), the document (e.g., the in-link count) and their relationship (e.g., the number of query terms in the document).

Since IR measures are not differentiable, their optimization is very challenging. To address this issue, the state-of-the-art solution is the LambdaRank gradient approximation [5], which is based on the idea of measuring the cost variation after swapping any two documents in a given result list. As discussed in [9], this approach can be applied to several IR measures and it is capable of accurately discovering local optima.

LambdaRank can be summarized as follows. d_i and d_j are two candidate documents for the same query q, with relevance labels l_i and l_j respectively, s_i and s_j are the currently predicted document scores. The lambda gradient of any given IR quality function Q is:

$$\lambda_{ij} = \frac{\partial Q_{ij}}{\partial \left(s_i - s_j\right)} \; = \; \mathrm{sgn}(y_i - y_j) \left| \Delta Q_{ij} \cdot \frac{1}{1 + e^{s_i - s_j}} \right|$$

where, the sign is determined by the document labels only, the first factor ΔQ is the quality variation when swapping scores s_i and s_j, and the second factor is the derivative of the RankNet cost [3], which minimizes the number of disordered pairs. When $l_i \geq l_j$, the quality Q increases with the score of document d_i. The larger the quality variation ΔQ, the higher the document d_i should be scored. Note that the RankNet multiplier fades ΔQ if documents are scored correctly, i.e. $s_i \geq s_j$, and boosts ΔQ otherwise. The lambda gradient for a document d_i is computed by marginalizing over all possible pairs in the result list: $\lambda_i = \sum_j \lambda_{ij}$. LambdaRank uses *Normalized Discounted Cumulated Gain (nDCG)* as Q and so ΔQ is the variation in nDCG caused by the swap of two documents.

We enhance the existing LambdaMart algorithm by replacing the above Q with a new quality measure which integrates the proposed user dynamic δ. This new measure is called *Normalized Markov Cumulated Gain (nMCG)* and it is defined as follows:

$$\mathrm{nMCG@}k = \frac{\sum_{i \leq k} \left(2^{l_i} - 1\right) \cdot \delta^c(i)}{\sum_{h \leq k, \text{sorted by } \mathrm{l_h}} \left(2^{l_h} - 1\right) \cdot \delta^c(h)}$$

where l_i is the relevance label of the i-th ranked document and $\delta^c(i)$ is the user dynamic function at rank i relative to the query class c, either navigational or informational. Basically, nMCG can be seen as an extension of nDCG where the discount function is defined by the user dynamic and depends on the query class. Moreover, since δ^c depends on the query class, i.e. depends on the query q, we are optimizing two different variants of the same quality measure nMCG across the training dataset. Finally, $\Delta\mathrm{nMCG}_{ij}$ can be computed efficiently as follows:

$$\Delta\mathrm{nMCG}_{ij} = \frac{-\left(2^{l_i} - 2^{l_j}\right)\left(\delta^c(i) - \delta^c(j)\right)}{\sum_{h \leq k, \text{sorted by } \mathrm{l_h}} \left(2^{l_h} - 1\right) \cdot \delta^c(h)}.$$

Hereinafter, we use nMCG-MART to refer to the described variant of LambdaMart aimed at maximizing nMCG.

Note that the query class is known at training time, and therefore the algorithm can optimize the proper user dynamic δ^c. Nor the document relevance, neither the query class information are available at test time, therefore the algorithm should, at the same time, classify queries and rank documents according to the different class-based dynamics δ^c.

3 EXPERIMENTS

3.1 Experimental Setup

We remark that there is no publicly available dataset providing user session data, document relevance and query-document pairs features at the same time. Therefore, we have to use two different datasets: the first for the user dynamic derivation and the second for the LtR analysis.

We calibrate the proposed user model on the basis of the click log dataset provided by Yandex [17] (http://imat-relpred.yandex.ru/en/). The dataset is composed of 340,796,067 records with 30,717,251 unique queries, retrieving 10 URLs each. We used the training set, which consists of 5191 assessed queries with binary judgments, corresponding to 30,741,907 records. Notice that 9% of the sessions corresponds to navigational queries while the remaining 91% corresponds to informational ones.

The accuracy of the proposed algorithm is evaluated on three public LtR datasets, MSLR-WEB30K and MSLR-WEB10K, provided by Microsoft [15] and Istella provided by Tiscali Istella Web search engine [8]. Dataset MSLR-WEB30K encompasses 31,531 queries from the Microsoft Bing search engine for a total of 3,771,125 query-document pairs represented by 136 features. The dataset is provided as a 5-fold split. The MSLR-WEB10K dataset contains 10,000 queries samples at random from the previous. Dataset Istella provides 33,018 queries for a total of 3,408,630 query-document pairs represented by 220 features. The dataset is provided as a 60/20/20 train/validation/test split.

Both the Microsoft and Istella datasets use integer relevance labels in the range $[0, 4]$. In order to classify queries as navigational or informational we adopt the following criterion. A query is considered as navigational if it contains only one result with relevance label ≥ 3. Approximatively 15% of the queries in the Microsoft datasets are classified according to this heuristic as navigational queries, which is quite similar to the value measured on the Yandex dataset. The Istella dataset instead contains a smaller set of navigational queries, covering about 3% of the dataset.

Table 1: nDCG@10 and nMCG@10 across test datasets for different model sizes (results are averaged across the 5 folds for Microsoft datasets.). Statistically significant differences at $p = 0.05$ and at $p = 0.01$ w.r.t. λ-MART marked resp. with * and **.

	MSLR-WEB30K			MSLR-WEB10K			Istella-S		
	100	500	Full	100	500	Full	100	500	Full
Algorithm	nDCG@10								
λ-MART	0.4564	0.4759	0.4793	0.4479	0.4637	0.4634	0.7031	0.7451	0.7536
nMCG-MART	0.4598^{**}	0.4778^{**}	0.4808^{**}	0.4499^{**}	0.4646	0.4648^{*}	0.7070^{**}	0.7466^{*}	0.7549
Algorithm	nMCG@10								
λ-MART	0.4684	0.4878	0.4914	0.4609	0.4767	0.4768	0.7551	0.7970	0.8059
nMCG-MART	0.4718^{**}	0.4898^{**}	0.4933^{**}	0.4626^{*}	0.4782	0.4790^{**}	0.7595^{**}	0.8000^{**}	0.8090^{**}

3.2 Experimental Results

We compare the effectiveness of state-of-the-art LtR algorithm λ-MART and nMCG-MART both in terms of nDCG@10 and nMCG@10 metrics. Recall that, at training time, λ-MART optimizes nDCG while nMCG-MART exploits the proposed nMCG metric. The algorithms' hyper-parameters were set after parameter sweeping, similarly to [6], to a learning rate of 0.05, maximum number of leaves of 64, and a maximum number of trees of 1500. The actual number of trees is tuned on the validation set. We also evaluate smaller models with 100 and 500 trees. In Table 1 we report the effectiveness scores.

We first observe that nMCG-MART is more effective in optimizing nMCG in every dataset and with every model size. This was expected as the proposed algorithm is the only one aimed at optimizing the proposed nMCG. At the same time, this confirms the soundness of the integration of nMCG into LambdaMart.

An interesting result is that nMCG-MART always provides higher nDCG@10 than LambdaMart. Recall that even relative improvements in nDCG below 1% are significant in terms of user satisfaction [7]. According to randomization test, the improvement is statistically significant at $p = 0.01$ on the larger MSLR-WEB30K dataset and on the other datasets limited to the small models with 100 trees. The proposed nMCG seems to provide more stable results, as optimizing nMCG also helps in optimizing nDCG. We believe that nMCG@k is somehow a simpler function to maximize: for informational queries it mainly discriminates between documents inside and outside the top-k results, and for navigational queries an additional boost is given if the relevant document is ranked first. This possibly drives the learning algorithm along a smoother cost function. The benefit is larger at the initial training iterations as suggested by the statistically significant improvements on small models with 100 trees. Larger models reach a plateau of effectiveness where it is anyway difficult to improve further. These hypotheses needs a detailed investigation as part of our future work.

We conclude that the proposed nMCG may provide a better modeling of the user behavior and that it may also provide high quality rankings according to other quality metrics of interest.

4 CONCLUSION AND FUTURE WORK

In this paper we presented a way to describe user dynamic through a model based on Markov chains and we integrated this dynamic in LambdaMart by defining a new quality measure called nMCG. Moreover, since nMCG depends on the query type, the proposed algorithm optimizes two different versions of the same quality measure. Experiments conducted on publicly available datasets showed that the proposed algorithm improves over the state-of-the-art with respect to both nDCG and nMCG.

As future work we aim at analyzing the properties of nMCG as well as the correlations with other evaluation measures. Moreover, we will conduct a user study in order to investigate whether the metric correlates with the quality of a ranking perceived by a user.

Acknowledgments. This work was partially supported by the EC H2020 Program INFRAIA-1-2014-2015 SoBigData: Social Mining & Big Data Ecosystem (654024), and SID16 Ferro, PRAT 2016: Improving Information Retrieval Effectiveness via Markovian User Models.

REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving Web Search Ranking by Incorporating User Behavior Information. In SIGIR, pages 19–26, ACM, 2006.
[2] A. Broder. A Taxonomy of Web Search. *SIGIR Forum*, 36(2):3–10, 2002.
[3] C. J. C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to Rank Using Gradient Descent. In ICML, pages 89–96, ACM, 2005.
[4] C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An Overview. Technical Report, 2010.
[5] C. J. C. Burges, R. Ragno, and Q. V. Le. Learning to Rank with Nonsmooth Cost Functions. In NIPS, Vol. 6, pages 193–200, 2006.
[6] G. Capannini, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego and N. Tonellotto. Quality versus Efficiency in Document Scoring with Learning-to-Rank Models. In IPM, 52(6):1161–1177, 2016.
[7] O. Chapelle, T. Joachims, F. Radlinski, and Y. Yue. Large-scale Validation and Analysis of Interleaved Search Evaluation. In TOIS, 30(1):6:1–6:41, 2012.
[8] D. Dato, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. Fast Ranking with Additive Ensembles of Oblivious and Non-Oblivious Regression Trees. In TOIS, 35(2):15:1–15:31, 2016.
[9] P. Donmez, K. M. Svore, and C. J. C. Burges. On the Local Optimality of LambdaRank. In SIGIR, pages 460–467, ACM, 2009.
[10] M. Ferrante, N. Ferro, and M. Maistro. Injecting User Models and Time into Precision via Markov Chains. In SIGIR, pages 597–606, ACM, 2014.
[11] K. Hofmann, A. Schuth, S. Whiteson, and M. de Rijke. Reusing Historical Interaction Data for Faster Online Learning to Rank for IR. In WSDM, pages 183–192, ACM, 2013.
[12] T. Liu. Learning to Rank for Information Retrieval. IN FnTIR, 3(3):225–331, 2009.
[13] C. Lucchese, S. Orlando, R. Perego, F. Silvestri and G. Tolomei. Discovering tasks from search engine query logs. In TOIS, 31(3):14:1–14:43, 2013.
[14] J. R. Norris. Markov Chains. Cambridge University Press, 1998.
[15] T. Qin, and T. Liu. Introducing LETOR 4.0 Datasets. In CoRR, 2013.
[16] A. Schuth, K. Hofmann, S. Whiteson, and M. de Rijke. Lerot: An Online Learning to Rank Framework. In LivingLab, pages 23–26, ACM, 2013.
[17] P. Serdyukov, N. Craswell, G. Dupret. WSCD2012: Workshop on Web Search Click Data 2012. In WSDM, pages 771–772, ACM, 2012.
[18] I. Teodorescu. Maximum Likelihood Estimation for Markov Chains. In arXiv preprint arXiv:0905.4131, 2009.
[19] Q. Wu, Burges, C. J. C. Christopher K. M. Svore, J. Gao. Adapting boosting for information retrieval measures. In Information Retrieval, 13(3):254–270, 2010.
[20] E. Yilmaz, M. Shokouhi, N. Craswell, and S. Robertson. Expected Browsing Utility for Web Search Evaluation. In CIKM, pages 1561–1565, ACM, 2010.
[21] Y. Zhang, L. A. F. Park, and A. Moffat. Click-based Evidence for Decaying Weight Distributions in Search Effectiveness Metrics. In Information Retrieval, 13(1):46–69, 2010.