Learning to Rank with Multiple Objective Functions

Krysta M. Svore
Microsoft Research
One Microsoft Way
Redmond, WA 98052
ksvore@microsoft.com

Maksims N. Volkovs
University of Toronto
40 St. George Street
Toronto, ON M5S 2E4
mvolkovs@cs.toronto.edu

Christopher J. C. Burges
Microsoft Research
One Microsoft Way
Redmond, WA 98052
cburges@microsoft.com

ABSTRACT

We investigate the problem of learning to rank for document retrieval from the perspective of learning with multiple objective functions. We present solutions to two open problems in learning to rank: first, we show how multiple measures can be combined into a single *graded measure* that can be learned. This solves the problem of learning from a 'scorecard' of measures by making such scorecards comparable, and we show results where a standard web relevance measure (NDCG) is used for the top-tier measure, and a relevance measure derived from click data is used for the second-tier measure; the second-tier measure is shown to significantly improve while leaving the top-tier measure largely unchanged. Second, we note that the learning-to-rank problem can itself be viewed as changing as the ranking model learns: for example, early in learning, adjusting the rank of all documents can be advantageous, but later during training, it becomes more desirable to concentrate on correcting the top few documents for each query. We show how an analysis of these problems leads to an improved, iteration-dependent cost function that interpolates between a cost function that is more appropriate for early learning, with one that is more appropriate for late-stage learning. The approach results in a significant improvement in accuracy with the same size models. We investigate these ideas using LambdaMART, a state-of-the-art ranking algorithm.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Retrieval models*; I.2.6 [**Artificial Intelligence**]: Learning

General Terms

Algorithms, Experimentation, Theory

Keywords

Learning to Rank, Web Search, Relevance Measures

1. INTRODUCTION

Ranking is central to many information retrieval (IR) applications, such as web search, collaborative filtering, and document retrieval. Large data sizes make it impractical to manually adapt the parameters of the ranking functions. Thus, several methods [6, 4, 5, 17, 18, 19], have recently been developed to automatically learn the model parameters using machine learning techniques — a problem known as *learning to rank*. In this paper, we focus on web search.

Learning to rank algorithms typically use labeled data, for example, query-URL pairs that have been assigned one of several levels of relevance by human judges [5]. However, often there are several additional sources of relevance labels available. For example, in addition to human judgments, search engines today gather user behavior information, such as clicks and dwell time. Further examples include *freshness* (the recency of the data on the page, which can be computed automatically), *spamness* (the likelihood that the page is spam), and *grammaticality* (the quality of the written language and grammar). Having several sources of relevance labels per query-URL pair can compensate for weaknesses in each single source. For example, using human judgments to assign each query-URL pair to, say, one of five levels of relevance, leaves open the questions of how to rank items with the same label, and how well the judges can solve the difficult problem of gauging user intent; clearly user click data should be able to help answer these questions. A search engine that leverages such markedly distinct sources of relevance would have an advantage over one that doesn't; however, this immediately raises the questions of how to compare the quality of two sets of search results given a 'scorecard' rather than a single numeric measure, and how to train systems using multiple label sources. Last but not least, the builders of a commercial search engine that uses, for example, human judgments only, may be reluctant to switch to another, unproven evaluation measure computed over labels from a different source; in contrast, given the entailed risk and having a graded measure which assures that their familiar measure will reliably be left unimpacted (or improved), while simultaneously improving the unproven measure, could prove valuable.

In this paper we address these problems by introducing the notion of a *graded measure*.[1] Consider the case where three IR measures $\{A, B, C\}$ (each of which maps a given ordering of documents for a given query to a score) are available, all taking values in $\Re$. The graded measure is then simply the triplet $\mathcal{A} \equiv \{A, B, C\}$ with the associated lexicographic ordering, such that, for example, $\mathcal{A}_1 \geq \mathcal{A}_2$ iff $A_1 \geq A_2$, or $A_1 = A_2$ and $B_1 \geq B_2$, or $A_1 = A_2$ and $B_1 = B_2$ and $C_1 \geq C_2$. Clearly this definition extends to any number of real-valued measures (and associated label

[1]By analogy to the formal notion of a graded vector space.

WWW 2011, March 28–April 1, 2011, Hyderabad, India.
ACM 978-1-4503-0632-4/11/03.

sources), and any two values of a graded measure are comparable. Here we consider the following two measures as an example: A will be taken as Normalized Discounted Cumulative Gain [13], and B will be a click-based measure [10, 14, 16]. Another intuitive way to view this approach is as follows: if, for example, the test set contains 10,000 queries, then since the NDCG is a single number computed as the mean NDCG over all queries, it seems reasonable to expect that one could keep the same NDCG while improving a secondary measure, such as a click-based one.

We choose the second measure to be click-based because clicks can be collected at a much lower cost and provide direct user feedback regarding user intent for a large number of real-world query-URL pairs, compared to human-based labels gathered from a small number of annotators who may have different knowledge, background, and opinions about relevance. We also use clicks in the secondary rather than primary measure since clicks have a strong position bias [15] and a high degree of noise, even when the position bias is removed [14]. In this paper, we define a graded measure over {NDCG, CNDCG}, where CNDCG is a defined click-based measure, and propose learning the graded measure within the LambdaMART ranking algorithm [19], which has been shown to have excellent empirical accuracy, winning the recent Yahoo! Learning to Rank Challenge [8]. LambdaMART is particularly well-suited to learning IR measures, and a graded IR measure, since learning for such a measure only requires expressing the gradient of the cost with respect to the model scores, and not the cost itself.

Although the notion of using several sources of relevance is not new, for example learning from user feedback data as ranking features [1], or as labels to replace human labels [10], our work addresses an open learning-to-rank question regarding tiered learning from multiple label sources. Our approach learns from several sources of labels and from a graded, user-defined measure over those labels. Our approach is particularly appealing because there is typically a high degree of conflict between label sources, thus making the existence of the globally optimal ranking, which perfectly satisfies each source, very unlikely. The graded metric idea effectively avoids this problem by optimizing for each part of the graded measure based on the user-imposed constraints. Furthermore, our approach is general, allowing the measures, label sources, and degree of preference between them, to be specified.

Our work also differs from prior work on learning for multiple measures, where each measure employs the *same* label source, for example as in [7], where both NDCG and Mean Reciprocal Rank (MRR) are computed over human relevance labels and then simultaneously optimized using a structured SVM framework. To our knowledge, our work is the first to combine multiple objectives that learn from multiple label sources in a graded way, and that can take advantage of a rich understanding, coming from several data sources, of the importance of a training sample.

The second part of the paper reveals three problems with optimizing for a single measure using a static objective function, which will be described in more detail below, but which we summarize here as follows. First, search engine accuracy is typically evaluated based on the documents in the top few rank positions, since those are the documents most viewed by users; take, for example, NDCG truncated to the top three documents. In the spirit of optimizing for what one cares about, one might conclude that we should train using NDCG@3 as the target measure, since LambdaRank (of which LambdaMART is the boosted tree instantiation) can model any IR measure [9]. However, it has been shown that this gives suboptimal results, probably because the effective amount of training data drops precipitously by so doing [4, 9]. LambdaRank and LambdaMART are thus typically trained using un-truncated NDCG. In this paper, we examine the question: can we get the best of both worlds by training using un-truncated NDCG early during training and gradually using truncated NDCG during later stages of training? The key idea is to gradually modify the learning process as the learning progresses by shifting the emphasis on the full ranking to the top of the list. This approach allows us to utilize all of the training data while fine-tuning the model for the target evaluation measure. The second problem relates to the margin built into the LambdaRank cost. For a query for which all of the documents happen to be ranked correctly by the current model, the LambdaRank gradients (which can be thought of as forces on unit masses, where the masses represent the documents) are still non-zero: LambdaRank still tries to further separate the (correctly ranked) documents. The third problem is related: suppose that instead, late during training, a highly relevant document is incorrectly ranked very low in the list. LambdaRank will try hard to fix this (the force on that document will be large, and pushing upwards), when it likely cannot be fixed (for example, due to a noisy label). We give these last two problems the mnemonics *Don't Fix what Isn't Broken* and *Give Up on Lost Causes*. Intuitively, the hope is that freeing the ranker from having to spend its capacity learning to model these effects will improve generalization accuracy, or at a minimum achieve the same accuracy with smaller models.

In this paper, we address the above problems by considering multiple measures as training targets. In the graded measure case, the measure is composed of two standard measures (although the technique generalizes to any number of standard measures). In the other case, the target measure essentially becomes a linear combination of two measures, such as NDCG and NDCG@3, and an interpolation over iterations is performed between the two desired standard measures.

2. GENERAL FRAMEWORK

We begin by formalizing the problem of learning to rank in the document retrieval domain. At training time, we are given a set of N queries $Q = \{q_1, ..., q_N\}$. To simplify notation, we drop the query index i, and refer to a general query q. Each query q is associated with a set of K documents $D = \{d_1, ..., d_K\}$ and their associated relevance labels $L = \{\ell_1, ..., \ell_K\}$. Each document d_j is represented as a query dependent feature vector in $\Re^p$, where $d_j[v]$ denotes the v^{th} feature value, with a corresponding relevance label $\ell_j \in \Re$ (typically an integer) that indicates how relevant document d_j is to query q.

The goal of learning is to create a ranker F such that, given a set of documents D with relevance labels L for query q, the ranking of documents in D produced by F has maximal agreement with L. In this paper we concentrate on LambdaMART's model of F, where $F : \Re^p \rightarrow \Re$ is a document scoring function which assigns scores used for ranking.

We use $S = \{s_1, ..., s_K\}$ to denote the scores assigned by F to D.

We evaluate the agreement between the ranking produced by F and the relevance labels L for query q with Normalized Discounted Cumulative Gain (NDCG) [13]:

$$\text{NDCG@}T(q) = \frac{1}{Z} \sum_{t=1}^{T} \frac{2^{\ell(t)} - 1}{\log(1+t)}, \tag{1}$$

where the sum is over documents for query q in ranked top-down order, $\ell(t)$ is the label of the document at rank position t, and Z is a normalizing constant which guarantees that NDCG is between 0 and 1. The truncation T is a constant typically set to a small value to emphasize the importance of correctly ranking documents at the top list positions. We report mean NDCG — the average NDCG(q) across all queries, and drop q to simplify notation. NDCG is well-suited for information retrieval evaluation due to its handling of multilevel labels and its dependence on truncation level. The main challenges in optimizing NDCG during training are that it contains a truncated summation over ordered items and is therefore not smooth and not differentiable, and that T is generally set to be small, such as 3, which ignores changes in rank position below position 3. We address these challenges in this paper.

3. LAMBDAMART

Since our approach extends the LambdaMART framework [3, 19], in this section we briefly review the ranking algorithm. We choose to extend LambdaMART because it has shown excellent empirical accuracy, recently winning the Yahoo! Learning to Rank Challenge [8, 2], and requires only the definition of the desired gradients of a cost function, which avoids the problem of differentiating a measure that requires document sorting, as in most IR measures, which are either flat, or discontinuous, everywhere. LambdaMART is a combination of the tree-boosting optimization method MART [11] and the listwise ranking model LambdaRank [4]. LambdaMART learns pairwise preferences over documents with emphasis derived from the NDCG gain found by swapping the rank position of the documents in any given pair, so it is a listwise algorithm (in the sense that the cost depends on the sorted list of documents). Remarkably, LambdaRank has been shown empirically to find local optima in NDCG [9].

We now describe the objective function and gradients used in training both LambdaRank and LambdaMART. The RankNet objective function, on which the LambdaRank objective is based, is a pairwise cross-entropy cost applied to the logistic function of the difference of the model scores [5]. Formally, given a pair of documents d_j, d_k and assuming $\ell_j > \ell_k$, then the objective is expressed as:

$$O_{jk} \equiv O(o_{jk}) = -o_{jk} + \log(1 + e^{o_{jk}}), \tag{2}$$

where $o_{jk} \equiv s_j - s_k$ is the score difference. The derivative of the objective according to the score difference is

$$\partial O_{jk}/\partial o_{jk} = \partial O_{jk}/\partial s_j = -1/(1 + e^{o_{jk}}). \tag{3}$$

The LambdaRank framework uses a smooth approximation to the gradient of a target evaluation measure with respect to the score of a document at position j, and we denote the λ-*gradient* as λ_j. The λ-gradient to best optimize for NDCG (based on empirical evaluation) is defined as the derivative of the objective (Eq 3) weighted by the difference in NDCG obtained when a pair of documents swap rank positions:

$$\lambda_{jk} \equiv \left| \Delta\text{NDCG@}T(s_j, s_k) \frac{\partial O_{jk}}{\partial o_{jk}} \right|. \tag{4}$$

Thus, at the beginning of each iteration, the documents are sorted according to their *current* scores, and the difference in NDCG is computed for each pair of documents by keeping the ranks of all of the other documents constant and swapping only the rank positions for that pair. The change in NDCG is computed as

$$\begin{aligned} &\Delta\text{NDCG@}T(s_j, s_k) \\ &= N \left(2^{\ell_j} - 2^{\ell_k}\right) \left(\frac{1}{\log(1 + t(d_j))} - \frac{1}{\log(1 + t(d_k))} \right). \end{aligned} \tag{5}$$

The λ-gradient for document d_j is computed by summing the λ's for all pairs of documents (d_j, d_k) for query q:

$$\lambda_j = \sum_{\substack{k \in (d_j, d_k): \\ \ell_j \neq \ell_k}} \lambda_{jk}. \tag{6}$$

It should be noted that this λ representation is schematic, in two senses: first, LambdaRank (and hence LambdaMART) is defined by noting that the models used (neural nets and gradient boosted trees) only need the gradients of the cost with respect to the model scores to be specified, not the actual costs, and the λ's are then simply defined to behave smoothly as documents swap rank positions. Second, $|\Delta\text{NDCG@}T(s_j, s_k)|$ depends on the order of the documents sorted according to the current model's scores, and in this sense, by defining the gradients after the documents have been sorted by score (Eq 4), we avoid having to write a very complex objective function that would have to encapsulate the effect of the sort. We refer the reader to [3] for details.

The $|\Delta\text{NDCG}|$ factor emphasizes those pairs that have the largest impact on NDCG. Note that the truncation in NDCG is relaxed to the entire document set to maximize the use of the available training data [9]. To optimize for NDCG, LambdaMART implements the LambdaRank idea using MART, a boosted tree model where each tree models the derivatives of the cost with respect to the model scores for each training point, and in this case, those derivatives are the λ's. The scoring function that LambdaMART produces after M iterations can be written as

$$F(d_j) = \sum_{m=1}^{M} \alpha_m f_m(d_j), \tag{7}$$

where each $f_i(d_j) \in \Re$ is modeled by a single regression tree and $\alpha_i \in \Re$ is the weight associated with each regression tree. At each iteration the regression tree is created by first computing the λ_j's and their derivatives (ρ_j's):

$$\rho_j = \frac{\partial \lambda_j}{\partial o_j}. \tag{8}$$

The tree is then initialized at the root node and we loop through all documents to find the feature v and the threshold ξ such that, if all document with $d_j[v] \leq \xi$ fall to the left child node, and the rest to the right child node, then the sum

$$\sum_{j \in \mathcal{L}} \left(\lambda_j - \bar{\lambda}_{\mathcal{L}}\right)^2 + \sum_{j \in \mathcal{R}} \left(\lambda_j - \bar{\lambda}_{\mathcal{R}}\right)^2$$

is minimized. Here $\mathcal{L}$ ($\mathcal{R}$) is the set of indicies of documents that fall to the left (right) subtree and $\bar{\lambda}_{\mathcal{L}}$ ($\bar{\lambda}_{\mathcal{R}}$) is the mean of the λ's for the set of samples that fall to the left (right). The split is attached to the root node and the process is repeated at each child node. Each f_i thus maps a given d_j to a real value by passing d_j down the tree, where the path (left or right) at a given node in the tree is determined by the value of a particular feature of d_j and where the output of the tree is taken to be a fixed value associated with each leaf ℓ:

$$\gamma_k = \frac{-\sum_{d_j \in \Phi_\ell} \lambda_j}{\sum_{d_j \in \Phi_\ell} \rho_j},$$

where Φ_ℓ is a set of documents that fall into leaf ℓ. The denominator here corresponds to taking a Newton step. By adding ρ to the documents' scores we effectively take a step in the direction that minimizes the objective function. A global multiplicative learning rate is often also used. Note that the α_i can be viewed as all taking the value one, since each leaf value combines the gradient, Newton step, and learning rate into one number. Below, it will guide intuition to note that the λ's can be interpreted as forces on point masses (represented by the documents), pushing the documents up or down the list [4].

The LambdaMART approach is very general: it extends the MART model beyond handling costs for which gradients are defined and well-behaved, to IR measures for which they are not. However it does not naturally handle multiple sources of relevance. In Section 4 we show how the lexicographic measure described above can be adapted in the LambdaMART framework to handle graded measures, and we apply it to a tiered measure of human judgments and clicks.

4. MULTIPLE SOURCES OF RELEVANCE

As mentioned above, relevance labels derived from human judges have several disadvantages, and recent work has explored alternative sources, such as user click-through rates [10, 14, 12, 16]. However, click-through rates also have several significant drawbacks, such as a high degree of noise and a strong position bias. In this paper we address this issue by offering a solution that allows optimization over multiple measures that are *graded*. We assume the first-tier measure is NDCG and the second-tier measure is a click-based measure, however our framework is general and can be extended to other measures and other sources of relevance labels such as freshness or grammaticality. A key advantage of our approach is that it allows for a graded combination of, for example, a potentially more trusted label (e.g., human judge) with a potentially noisy or "yet to be trusted as the sole label source" label (e.g., click information). We begin by defining one possible target evaluation measure for clicks (other examples include [12, 16]).

Here we assume that the labels L come from human judgments and take values in $\{0, 1, 2, 3, 4\}$, where 0 indicates a bad and 4 indicates a perfect document for the query. We also assume that we have a second label for query q denoted by $C = \{c_1, ..., c_m\}$, where in our work we assume that the secondary label source is the query click logs, and each $c_j \in [0, 1]$ is a score derived from a given function of the query click information for document d_j. However, many possible label sources can be considered for both the primary and secondary labels. We use superscripts L and C to denote that the labels being used are from label source L or C, respectively, i.e., $\lambda_j^{(L)}$ and $O^{(L)}$.

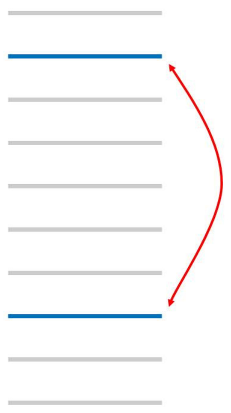


Figure 1: A set of documents ordered for a given query. The light gray bars represent irrelevant (bad) documents, while the dark blue bars represent relevant (good) documents. NDCG@3 will not be affected if the two relevant documents are swapped, as shown by the red arrow. However, if the currently lower-ranked relevant document is clicked on more than the higher ranked one (has a higher click label), then the swap will improve CNDCG@3 without affecting NDCG@3.

4.1 Click Evaluation Measure

To evaluate a ranking model on clicks, we need a suitable evaluation measure. Several measures have been proposed in the literature [10, 14, 16, 12]; a popular one is Kendall's τ [10, 14], which compares pairwise preferences induced by clicks with those induced by the ranking model's output scores. Kendall's τ is well suited to comparing full rankings, but is less desirable when preferring the top of the list. Our aim is to emphasize the accuracy at the top of the ranked list, and to ultimately optimize for a combination of measures. Thus for simplicity, we choose to adapt NDCG, a measure that more heavily rewards correctly ranking the top of the list, to click-based relevance labels as follows:

$$\text{CNDCG@}T = \frac{1}{N} \sum_{t=1}^{T} \frac{2^{c(t)\times 4} - 1}{\log(1+t)}, \qquad (9)$$

where the click label $c(t) \in [0, 1]$ is a function of click information for the document at rank position t, and multiplying by 4 (the maximum value of labels L) extends the numerator to the same range as the human-derived relevance labels L, making CNDCG scores comparable to NDCG scores.

4.2 Learning a Graded Measure

Our aim is to improve CNDCG without decreasing NDCG, in particular at truncation level three. For a graded measure, one ranking model is to be preferred over another if it gives at least equivalent NDCG scores, with improved CDCG scores, on a test (valid) set. We explore this in the listwise framework of LambdaMART by requiring that document pairs with the same human relevance label ℓ should be swapped if the lower-ranked document has a higher click label c (higher user click-through rate). The intuition behind our approach is illustrated in Figure 1. To achieve this requirement, we define a λ-gradient $\lambda^{(C)}$ that is designed to optimize for CNDCG and considers click labels c_j, c_k for pairs of documents (d_j, d_k) with the same human relevance label $\ell_j = \ell_k$:

$$\begin{aligned}\lambda_j^{(C)} &= \sum_{\substack{k \in (d_j, d_k): \\ c_j > c_k \\ \ell_j = \ell_k}} \lambda_{jk}^{(C)} \\ &= \sum_{\substack{k \in (d_j, d_k): \\ c_j > c_k \\ \ell_j = \ell_k}} |\Delta \text{CNDCG@}T(s_j, s_k)| \log(1 + e^{s_k - s_j}),\end{aligned}$$

where $|\Delta\text{CNDCG@}T(s_j, s_k)|$ is the difference in CNDCG obtained when d_j is swapped with document d_k in the current ranking. CNDCG prefers rankings where for every pair of documents with the same relevance label, the document that is clicked more is ranked higher. Furthermore, note that $\lambda_j^{(L)}$ given in Eq 6 does not include pairs of documents with the same relevance label, so the learning signal provided by $\lambda_j^{(L)}$ does not contradict that provided by $\lambda_j^{(C)}$. To optimize for both NDCG and CNDCG simultaneously, we linearly combine the λ-gradients:

$$\lambda_j = (1 - w)\ \lambda_j^{(L)} + w\ \lambda_j^{(C)}, \qquad (10)$$

where $w \in [0, 1]$ controls the contribution of each λ-gradient, allowing the tiering of measures to be in a sense more or less graded. To minimize conflicting signals between the two λ's, we exclude documents with zero clicks from $\lambda_j^{(C)}$, primarily because zero clicks can result from users not finding a document relevant, or from users not seeing a document, which can happen if the document is ranked further down the list, or if the document has never been ranked. In either case, the document could still be highly relevant, making it potentially dangerous to treat zero clicks as an indicator of irrelevance.

To investigate the agreement between the two λ-gradients, we trained a model to optimize for NDCG by using $\lambda_j^{(L)}$ during learning, but for comparison purposes computed both $\lambda_j^{(L)}$ and $\lambda_j^{(C)}$ at each iteration. We define λ-gradient agreement to be either when the signs of $\lambda_j^{(L)}$ and $\lambda_j^{(C)}$ for document d_j were the same or when $\lambda_j^{(C)} = 0$. Figure 2 shows the percent of λ's that agree as learning progresses for two cases: (1) $\lambda_j^{(C)}$ is computed over all $\{(d_j, d_k) : c_j > c_k\}$ pairs and (2) $\lambda_j^{(C)}$ is computed over pairs $\{(d_j, d_k) : c_j \neq 0,\ c_k \neq 0,\ c_j > c_k, \ell_j = \ell_k\}$. When all click pairs are used (case 1), the agreement between the two λ signals is roughly 70% (ignoring the first iteration). This indicates that the two λ-gradient signals are relatively compatible, and that it should be possible to optimize for both NDCG and CNDCG simultaneously. Furthermore, restricting the use of $\lambda_j^{(C)}$ to pairs of documents with the same human relevance label and removing documents with zero clicks significantly improves agreement by almost 15 percentage points, indicating that such a restriction during learning could offer additional improvements in accuracy. Section 6 details our results on using a linear combination of λ-gradients to learn a graded measure.

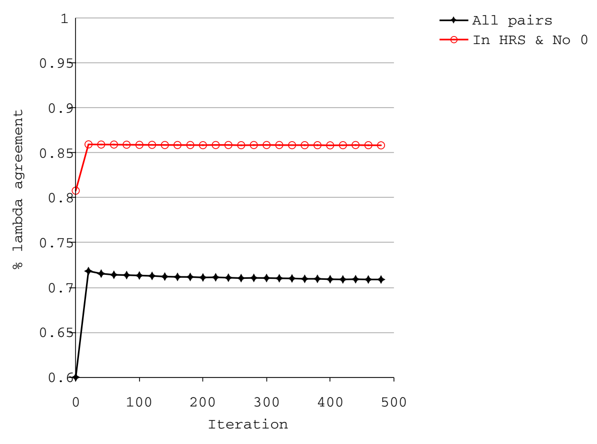


Figure 2: Training iteration versus percent λ agreement on training data when $\lambda_j^{(C)}$ uses all $\{(d_j, d_k) : c_j > c_k\}$ pairs (black diamonds) versus $\{(d_j, d_k) : c_j \neq 0, c_k \neq 0, c_j > c_k, \ell_j = \ell_k\}$ pairs (red circles).

5. OPTIMIZING FOR POSITION THREE

We now consider learning with several λ-gradients to optimize for NDCG and NDCG@3, using a single label source. Despite the emphasis on truncation three, LambdaMART utilizes all of the training data by relaxing the truncation for NDCG in $\lambda_j^{(L)}$ to include all documents. In this section, we investigate the effects of this relaxation on learning and propose a modified λ-gradient designed to better optimize for NDCG@3. We use NDCG@3 here for illustrative purposes; a similar analysis can easily be extended to NDCG at other truncations.

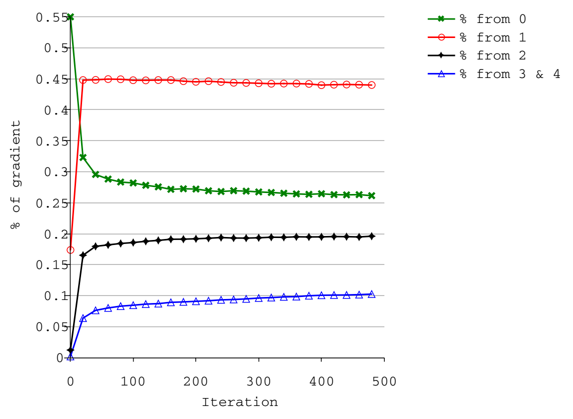
5.1 Effects of $\lambda^{(L)}$ on Learning NDCG

The document gradients $\lambda_j^{(L)}$ govern how the ranking model changes with each iteration of learning. To gain insight, we monitored $\lambda^{(L)}$'s throughout learning. In Figure 3(a), for each highly relevant document d_j with $\ell_j = \{3, 4\}$ in our training set, we plot the following percentage

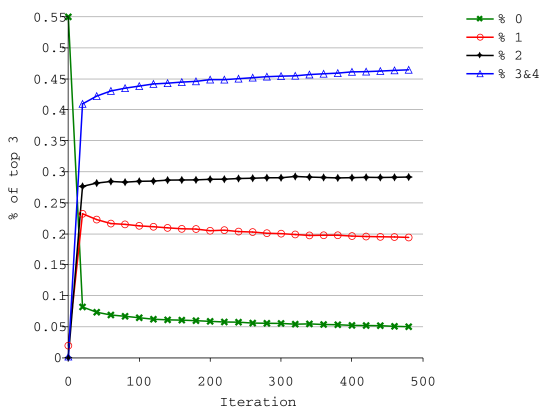
$$\frac{1}{H} \sum_{j:\ell_j = \{3,4\}} \frac{\sum_{\ell_k = v} \lambda_{jk}^{(L)}}{\lambda_j^{(L)}}, \qquad (11)$$

for $v = \{0, 1, 2, 3 \text{ and } 4\}$, and where H represents the number of highly relevant documents across all queries. Throughout learning, contributions from irrelevant and weakly relevant documents ($\ell_k = \{0, 1\}$) consistently account for on average more than 70% of the total gradient value of a highly relevant document. During the initial stages of learning, it is desirable to strongly separate highly relevant documents from irrelevant documents. During later stages, however, most irrelevant documents will be already low in the ranked list, and the NDCG@3 gains will come not from further separating relevant and irrelevant documents, but from fine-grained separations among the top 3 positions. That is, separations between documents with labels 3 and 4, or 2 and 3, for example, become more and more essential as learning progresses.

To support this hypothesis, we plot the distribution of the



(a)



(b)

Figure 3: Figure 3(a) shows percent λ contribution to highly relevant documents with relevance labels 3 and 4 from all relevance labels. Figure 3(b) shows the distribution of relevance labels in the top 3 rank positions throughout the 500 learning iterations, averaged across all training queries.

label values ℓ_j for documents d_j ranked in the top 3 positions using the model at iteration t as learning progresses, as shown in Figure 3(b). After only a few iterations, irrelevant documents ($\ell_j = 0$) account for less than 5% of the documents in the top 3 positions. Furthermore, weakly relevant documents ($\ell_j = 1$) account for less than 20% of the documents in the top 3 positions. Over 40% of documents are highly relevant within the first few iterations, further demonstrating that the learning algorithm is able to almost immediately separate highly relevant documents from irrelevant ones, and that encouraging fine-grained separations among top-ranked documents may lead to more accurate ranking models.

Formally, irrelevant documents low in the ranked list dominate the λ-gradient values of highly relevant documents throughout learning due to the structure of the $\lambda^{(L)}$'s. Recall that $\lambda_{jk}^{(L)}$ is a sigmoid weighted by the change in NDCG when documents d_j and d_k swap rank positions (Eq 4). When the pair of documents is ranked correctly, $o_{jk} = s_j - s_k > 0$, the sigmoid will approach 0, but the ΔNDCG component will grow with increasing rank position, as illustrated by the following example. Suppose there are 3 documents: d_1 ($\ell_1 = 4$) in position 1, d_2 ($\ell_2 = 0$) in position 4, and d_3 ($\ell_3 = 0$) in position 50. The ΔNDCG values for the corresponding pairs, namely (d_1, d_2) and (d_1, d_3), are:

$$|\Delta\text{NDCG}(s_1, s_2)| = \frac{(2^4 - 2^0)}{N}\left(\frac{1}{\log(2)} - \frac{1}{\log(4)}\right) = \frac{10.8}{N},$$
$$|\Delta\text{NDCG}(s_1, s_3)| = \frac{(2^4 - 2^0)}{N}\left(\frac{1}{\log(2)} - \frac{1}{\log(50)}\right) = \frac{17.8}{N}.$$

Pair (d_1, d_3) thus gets almost double the weight even though d_3 is far down the list. In many cases, this weight will counteract the decreasing sigmoid, causing $\lambda_{1,3}^{(L)}$ to significantly contribute to $\lambda_1^{(L)}$. In most learning-to-rank datasets there is also a significant label imbalance; the number of irrelevant documents typically far outnumber the number of relevant ones. This further magnifies the effects of the ΔNDCG component on learning, since the large number of irrelevant documents ranked low will additively dominate the λ-gradient.

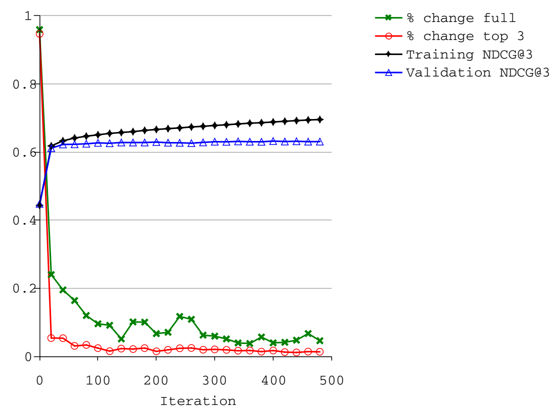


Figure 4: Train and validation NDCG@3, together with percent of changes that happen across the entire ranking and within the top-3 positions, averaged over all training queries.

A similar observation can be made for incorrectly ranked pairs. When the difference $o_{jk} < 0$ for incorrectly ranked documents d_j and d_k is large, the sigmoid asymptotes to 1. Now consider the previous example with swapped labels: d_1 ($\ell_1 = 0$) in position 1, d_2 ($\ell_2 = 4$) in position 4, and d_3 ($\ell_3 = 4$) in position 50. Because the difference in scores is larger for pair (d_1, d_3), the sigmoid will either be 1 for both (d_1, d_2) and (d_1, d_3), or it will be larger for (d_1, d_3). Swapping the labels has no effect on the $|\Delta\text{NDCG}|$, so (d_1, d_3) will still get almost double the weight. Thus, the model will learn to separate d_1 from d_3 more heavily than d_1 from d_2, when fixing (d_1, d_2) is much easier. The LambdaRank gradient appears to emphasize global changes in ranking throughout learning, instead of targeting easier, more local adjustments. In addition, focusing on global changes could be risky, in particular if, say, d_3 has an incorrect label.

More evidence that learning at the top of the list is unable to significantly improve can be seen from Figure 4, which shows training and validation NDCG@3, as well as the fraction of changes in ranking that occur in the top 3 positions versus across the entire ranking, averaged across all training queries. After 100 iterations, less than 3% of changes occur in the top 3 positions, while there are between 5% and 15% rank changes across the whole list.

In summary, we have identified two related effects of $\lambda^{(L)}$ on learning:

1. $\lambda^{(L)}$ concentrates on separating irrelevant and highly relevant documents, thus *Fixing what isn't broken*,
2. $\lambda^{(L)}$ emphasizes large global changes in ranking over local ones, thus *Not giving up on lost causes*.

Since both of these effects arise in later stages of learning, our solution is to gradually modify the λ-gradients to encourage learning to correctly rank the top of the list. To make this transition smooth, we propose an *iteration-dependent* linear combination of two λ-gradients that adjusts the weight w_m with each boosting iteration m:

$$\lambda_m = (1 - w_m)\ \lambda^{(L)} + w_m\ \lambda^{(S)}, \tag{12}$$

where $\lambda^{(L)}$ is the LambdaRank gradient, $\lambda^{(S)}$ is a new gradient, and $w_m \in [0, 1]$. In the next section, we define $\lambda^{(S)}$, which aims to address the aforementioned challenges and optimize more directly for NDCG@3.

5.2 A Modified Objective and λ-gradient

The effects discussed above partially arise from the cross-entropy cost used in the objective function of LambdaRank and LambdaMART (Eq 2). The derivative of the cost is a sigmoid (Eq 3) and is plotted in Figure 5. As the difference in scores increases for an incorrectly ranked pair, the derivative approaches 1. Thus, pairs that are further apart have larger λ-gradients, causing the model to concentrate on fixing them. We thus need a cost whose derivative decreases as the difference in scores becomes significantly large. One choice is a sigmoid:

$$O^{(S)}_{jk} = \frac{1}{1 + \exp(s_j - s_k + \mu)}, \tag{13}$$

where μ is the center (mean) of the sigmoid. Differentiating this objective with respect to score difference (ignoring the sign), we get a weighted exponential:

$$\frac{\partial O^{(S)}_{jk}}{\partial o_{jk}} = \frac{\partial O^{(S)}_{jk}}{\partial s_j} = \frac{e^{o_{jk}+\mu}}{(1 + e^{o_{jk}+\mu})^2}, \tag{14}$$

also plotted in Figure 5. Note that when $o_{jk} > 0$, the derivative (Eq 14) drops off similarly to the derivative of $O^{(L)}$ (Eq 3). On the other hand, when $o_{jk} < 0$, Eq 14 asymptotes to zero as desired. Given a document with score s_j, the new function will have a neighborhood of $[s_j - \mu - \epsilon, s_j - \mu + \epsilon]$ where the derivative is non-zero, and the further we move from the center $s_j - \mu$ of this neighborhood, the smaller the gradient becomes. So this objective will strongly encourage local moves in ranking.

The corresponding λ-gradient $\lambda^{(S)}$ can be defined as follows:

$$\lambda^{(S)}_{jk} = |\Delta\text{NDCG@}T(s_j, s_k)| \frac{e^{o_{jk}+\mu}}{(1 + e^{o_{jk}+\mu})^2}. \tag{15}$$

However, we showed earlier that the $|\Delta\text{NDCG}|$ weight is larger for pairs of documents that are ranked further apart.

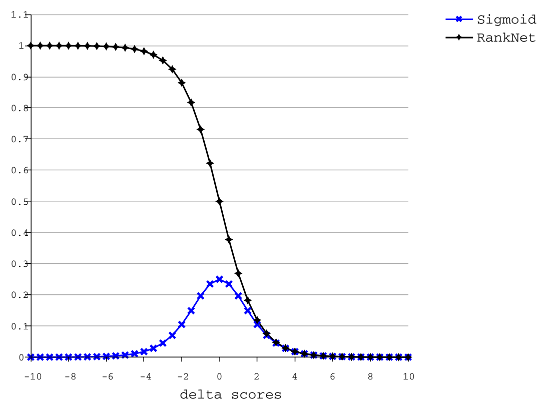


Figure 5: Plots of the derviatives of the two objectives: $O^{(L)}$ (Eq 3, black circle) and $O^{(C)}$ with $\mu = 0$ (Eq 14, blue square), both without the NDCG difference component. On the x-axis is the difference in scores o_{jk}; it is assumed that $\ell_j > \ell_k$ and document d_j should be ranked higher than document d_k.

This weight could counteract the signal from the the sigmoid cost and prefer documents far away from the center of the neighborhood. A potential way to address this problem is to raise the truncation level to 3 in the λ-gradient:

$$\lambda^{(S@3)}_{jk} = |\Delta\text{NDCG@3}(s_j, s_k)| \frac{e^{o_{jk}+\mu}}{(1 + e^{o_{jk}+\mu})^2}, \tag{16}$$

which discards pairs where both documents are ranked lower than 3; combining this with the sigmoid cost will result in non-zero gradients only for documents whose scores are close to scores of the documents in the top 3 positions. Below position 3, NDCG@3 no longer depends on the order of the documents. Using our example with d_1 ($\ell_1 = 0$) in position 1, d_2 ($\ell_2 = 4$) in position 4 and d_3 ($\ell_3 = 4$) in position 50, we now have that:

$$|\Delta\text{NDCG@3}(s_1, s_2)| = \frac{1}{N}\frac{2^4 - 2^0}{\log(2)} = \frac{21.6}{N},$$
$$|\Delta\text{NDCG@3}(s_1, s_3)| = \frac{1}{N}\frac{2^4 - 2^0}{\log(2)} = \frac{21.6}{N}.$$

Since the derivative of the sigmoid will be significantly larger for pair (d_1, d_2), the λ's will guide the model to swap d_1 and d_2. We have thus created an objective and corresponding λ-gradient which concentrates on the top of the ranked list and emphasizes local changes. Note, however, that using $\lambda^{(S@3)}$ may suffer from too few training pairs, thus it may still be advantageous to use $\lambda^{(S)}$ over $\lambda^{(S@3)}$.

5.3 Newton Step vs. Gradient Descent

LambdaMART uses the second derivative of the cost in a Newton step that scales each leaf value in each weak learner. With the new cost, however, the second derivative becomes

$$\rho^{(S)}_{jk} = |\Delta\text{NDCG@}T(s_j, s_k)| \frac{e^{o_{jk}+\mu}(e^{o_{jk}+\mu} - 1)}{(1 + e^{o_{jk}+\mu})^3}.$$

Since the difference in scores can be positive or negative, $e^{o_{jk}+\mu} - 1$ can be positive or negative; this makes it difficult to control the sign of the gradients when the average step size is computed in each leaf. To avoid this problem, we replace the Newton step with standard gradient descent when training with the iteration-dependent λ (Eq 12). In order to preserve the scale-independence, we normalize the λ_j's for each query by dividing them by their standard deviation.

6. EXPERIMENTS

We conducted our experiments on a large real-world web dataset. The queries were sampled from the user query logs of a commercial search engine, along with roughly 150–200 corresponding URLs per query. We divided the data into train/valid/test sets of size 108K/13.5K/13.5K queries, respectively. Each query-URL pair has 860 features, a human relevance label $\ell \in \{0, \ldots, 4\}$, and two types of click labels $c \in [0, 1]$. The click label for a query-document pair q-d is assigned the click-through rate computed over many users, where a click is when a user clicks on d after entering either (1) a query equivalent to q, denoted by $=$, or (2) a query similar or equal to q, denoted by $\approx$. To reduce the position bias for the click labels, we used the last clicks (document which was clicked on last during the user session) to measure the click-through rate for each document. This method provides a better indication of relevance than aggregating all clicks for each document. Typically, the fact that the user stopped browsing after clicking on a particular document often indicates that s/he was satisfied with that document.

Several of the 860 features used for training were functions of click-through rate. Typically, learning from features that encode the target label assignments would result in a model that severely overfits to the training data, essentially memorizing the labels by learning only from those features. However, in this case we have two sets of target labels and have enforced the constraint that NDCG must not change significantly. This constraint forces the model to generalize because the trivial mapping from click features to labels would perform very poorly on the primary NDCG objective. Moreover, since click-through rates are available for many documents at test time, we want to leverage this information when making model predictions. We do, however, remove those features from the model that encode the label exactly or near exactly in order to learn a more general model.

For each experiment, we performed extensive parameter sweeps over the LambdaMART parameters and the parameters of our weighted λ-gradients. For each model, we set the number of leaves to 30, the number of boosting iterations to 1000, and swept two values of the learning rate (0.2, 0.5) and three values of the number of documents required to fall in a given leaf node (6000, 8000, 10000). We also swept the parameters of each weighted objective function, as described in the text below. The best parameters for each model were determined based on the best NDCG@3 accuracy on the validation set. Our sweeps were extensive, and showed that relative accuracy among the models was fairly robust to parameter settings. Results are reported on the held-out test set. We report statistical significance using a paired t-test at the 95% confidence level, where we calculate the accuracy differences between two models on each query in our 13.5K test set and then compute the standard error. We compare the standard error of the differences against the mean difference; the model is significantly better if the ratio of the mean to the standard error is greater than 1.96.

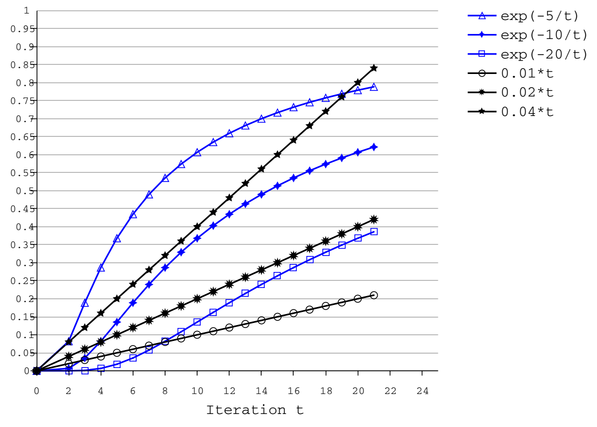


Figure 6: Examples of exponential and linear functions for three different values of η.

6.1 Results: Graded Objective

We first report results on learning to optimize for the graded measure using a graded λ-gradient (Eq 10), with NDCG (on labels L) as the tier-one measure and CNDCG (on labels C) as the tier-two measure. We swept over $w \in \{0.01, 0.1, 0.2, 0.3, 0.4, 0.5\}$. Table 1 shows results from training LambdaMART using $\lambda^{(L)}$ versus using the graded λ, for two different sources of click labels (with the Newton step). The results indicate that optimizing for the graded objective, in the case of both click label sources, yields equivalent NDCG scores to the baseline, while significantly improving CNDCG, at all truncation levels. Notably, the CNDCG scores increase by as much as 3–5 points at all truncations.

6.2 Results: Iteration-dependent Objective

We next report results on our iteration-dependent objective which uses an iteration-dependent λ-gradient (Eq 12), where the two objectives are the LambdaRank objective $O^{(L)}$ and the sigmoid objective $O^{(S)}$ (both use labels L). The starting weight w_0 was determined based on accuracy on the validation set, after sweeping over $w_0 \in \{0.1, 0.25, 0.5\}$. We experimented with two methods of increasing w_m, exponential and linear,

$$w_m = w_{m-1} + \exp(-\eta/m), \text{ and } w_m = w_{m-1} + \eta,$$

respectively, where the rate of increase η was chosen based on accuracy on the validation set, after sweeping over $\eta \in \{100, 250\}$ (exponential) and $\eta \in \{0.1, 0.01, 0.001\}$ (linear), respectively. Figure 6 shows examples of exponential and linear functions for different values of η as the numbers of boosting iterations increases.

The results for both linear and exponential methods are shown in Table 2. Firstly, it is notable that the baseline using gradient descent significantly outperforms the baseline using the Newton step, at all truncations except NDCG@1 and NDCG@2. Secondly, we find that using all truncation levels for learning, $\lambda^{(S)}$, in our iteration-dependent λ-gradient significantly beats using $\lambda^{(S@3)}$. Thirdly, training with the iteration-dependent λ based on $\lambda^{(L)}$ and $\lambda^{(S)}$ (denoted $\lambda^{(L)} + \lambda^{(S)}$) is significantly better than both the gradient-descent baseline and the Newton-step baseline at all truncations except NDCG@1 and NDCG@2. Finally, re-

Table 1: Results on LambdaMART (baseline) versus LambdaMART using the graded objective. Two types of click labels are considered: = and ≈. LambdaMART is trained using the Newton step (N.) in both cases. Bold indicates statistical significance of the graded objective over the baseline at the 95% confidence level.

Click	Method	Measure	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10
=	Baseline (N.)	NDCG	62.27	61.43	61.66	62.29	62.88	63.49	64.00	64.47	64.83	65.17
	$\lambda^{(L)} + \lambda^{(C)}$ (N.)	NDCG	62.49	61.47	61.77	62.32	62.97	63.49	63.96	64.42	64.78	65.15
	Baseline (N.)	CNDCG	38.54	41.71	43.12	43.93	44.45	44.80	45.06	45.28	45.44	45.59
	$\lambda^{(L)} + \lambda^{(C)}$ (N.)	CNDCG	**43.11**	**45.74**	**46.75**	**47.33**	**47.66**	**47.89**	**48.06**	**48.19**	**48.30**	**48.37**
≈	Baseline (N.)	NDCG	62.27	61.43	61.66	62.29	62.88	63.49	64.00	64.47	64.83	65.17
	$\lambda^{(L)} + \lambda^{(C)}$(N.)	NDCG	62.36	61.38	61.59	62.17	62.80	63.35	63.85	64.32	64.72	65.07
	Baseline (N.)	CNDCG	48.81	53.01	55.23	56.67	57.66	58.37	58.92	59.33	59.71	60.02
	$\lambda^{(L)} + \lambda^{(C)}$ (N.)	CNDCG	**53.80**	**57.20**	**59.33**	**60.59**	**61.47**	**62.08**	**62.60**	**62.99**	**63.29**	**63.56**

Table 2: NDCG results on the 13.5K test set for LambdaMART (baseline; both Newton and gradient descent) versus LambdaMART using the iteration-dependent objective for two combinations: (1) $\lambda^{(L)}$ and $\lambda^{(S)}$ and (2) $\lambda^{(L)}$ and $\lambda^{(S@3)}$. Two methods of weight adjustment are considered: exponential and linear. Significant differences are indicated in bold (95% confidence level) and in italic (90% confidence level). Significance of the baseline gradient descent model is over the baseline Newton model. Significance of $\lambda^{(L)} + \lambda^{(S)}$ models is over the baseline gradient descent model. Significance of $\lambda^{(L)} + \lambda^{(S@3)}$ models is omitted since they are consistently worse than the $\lambda^{(L)} + \lambda^{(S)}$ models.

Method	@1	@2	@3	@4	@5	@6	@7	@8	@9	@10
Baseline (Newton)	62.27	61.43	61.66	62.29	62.88	63.49	64.00	64.47	64.83	65.17
Baseline (Gradient Descent)	62.47	61.53	**61.94**	**62.57**	**63.17**	**63.75**	**64.28**	**64.77**	**65.16**	**65.51**
$\lambda^{(L)} + \lambda^{(S)}$, Exponential	62.66	61.69	**62.15**	**62.74**	**63.39**	**63.98**	**64.50**	**64.96**	**65.37**	**65.71**
$\lambda^{(L)} + \lambda^{(S)}$, Linear	62.56	*61.82*	**62.19**	**62.78**	**63.33**	**63.97**	**64.47**	**64.91**	**65.32**	**65.68**
$\lambda^{(L)} + \lambda^{(S@3)}$, Exponential	62.23	61.5	61.75	62.3	62.9	63.4	63.87	64.3	64.67	65.03
$\lambda^{(L)} + \lambda^{(S@3)}$, Linear	62.3	61.64	61.82	62.43	62.99	63.51	64.02	64.45	64.82	65.14

sults indicate that the two methods of weight adjustment, exponential and linear, produce equivalent ranking models, with the exception that the linear adjustment additionally results in a significant (at the 90% level) gain at NDCG@2 over the gradient descent baseline.

We also conduct experiments to determine if in the presence of smaller amounts of training data (for example in emerging search markets, where less labeled training data is available), training LambdaMART using the iteration-dependent gradients offers accuracy improvements over the baseline. Figures 7(a)–7(c) show NDCG@1, 3, 10 results on the 13.5K test set from training on varying percentages of the 108K training data. The results indicate that with even small amounts of training data, the iteration-depedent objective yields significant improvements over the baseline methods, at various truncation levels. Results also indicate, as previously shown, that the LambdaMART baseline using gradient descent outperforms the LambdaMART baseline using the Newton step.

7. CONCLUSIONS

We have shown how to extend the λ family of functions and apply our approaches to a state-of-the-art ranker, LambdaMART, in two ways: first, we showed how graded measures can be learned using our λ functions. This is an increasingly important issue for commercial search engines, as they typically want to optimize for several evaluation measures simultaneously, and simple scorecards of measures are not easily comparable. Second, we adjusted the λ functions to solve two problems that we showed occur with the current LambdaMART λ-gradient: the ranking model should stop trying to further separate documents that are already correctly ordered and well-separated, as well as ranking mistakes that persist long into training. The application of these ideas, all of which are based on training with multiple measures, resulted in ranking models that gave significant improvements in ranking accuracy over the baseline state-of-the-art ranker LambdaMART.

Future work includes extending our multi-tiered learning to include other standard IR measures. One can also imagine, as discussed in the introduction, having a triplet of measures, or perhaps an entire scorecard of measures, and thus extending learning to several measures of interest. We would also like to determine ways to learn for multiple measures when not all of the training samples have labels from the various sources. For example, click information is readily available on many pages that lack a human judgment. Developing a principled approach to learning for multiple measures employing several (sometimes missing) label sources is a promising direction for future research.

8. REFERENCES

[1] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior

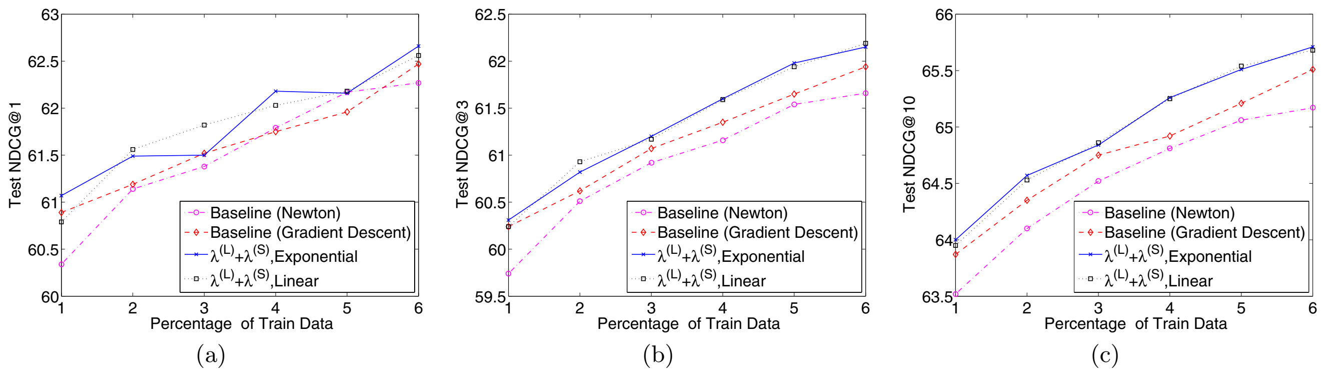


Figure 7: NDCG results on the 13.5K test set for LambdaMART (baseline; both Newton and gradient descent) versus LambdaMART using the iteration-dependent λ-gradient, denoted $\lambda^{(L)} + \lambda^{(S)}$. Two methods of weight adjustment are considered: exponential and linear. The x-axis indicates the percentage of the 108K training data used for training, with $\{1 = 3.25\%, 2 = 6.25\%, 3 = 12.5\%, 4 = 25\%, 5 = 50\%, 6 = 100\%\}$. The validation and test sets remain consistent across all experiments, each with 13.5K queries.

information. In *Proceedings of international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26, 2006.

[2] C. J. Burges, K. M. Svore, P. N. Benett, A. Pastusiak, and Q. Wu. Learning to rank using an ensemble of lambda-gradient models. *to appear in Special Edition of JMLR: Proceedings of the Yahoo! Learning to Rank Challenge*, 14:25–35, 2011.

[3] C. J. C. Burges. From RankNet to LambdaRank to LambdaMART: An Overview. Technical Report MSR-TR-2010-82, Microsoft Research, 2010.

[4] C. J. C. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Proceedings of the Neural Information Processing Systems*, pages 193–200, 2006.

[5] C. J. 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 International Conference on Machine learning*, pages 89–96, 2005.

[6] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the International Conference on Machine learning*, pages 129–136, 2007.

[7] S. Chakrabarti, R. Khanna, U. Sawant, and C. Bhattacharyya. Structured learning for non-smooth ranking losses. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 88–96, 2008.

[8] O. Chapelle, Y. Chang, and T.-Y. Liu. The Yahoo! Learning to Rank Challenge, 2010.

[9] P. Donmez, K. Svore, and C. Burges. On the local optimality of lambdarank. In *Special Interest Group on Information Retrieval (SIGIR)*, 2009.

[10] Z. Dou, R. Song, X. Yuan, and J.-R. Wen. Are click-through data adequate for learning web search rankings? In *Proceeding of the ACM Conference on Information and Knowledge Management*, 2008.

[11] J. H. Friedman. Greedy function approximation: A gradient boosting machine. In *Annals of Statistics*, pages 29(5):1189–1232, 2001.

[12] F. Guo, C. Liu, A. Kannan, T. Minka, M. Taylor, Y.-M. Wang, , and C. Faloutsos. Click chain model in web search. In *Proceedings of the 18th International World Wide Web Conference*. Association for Computing Machinery, Inc., 2009.

[13] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 41–48, 2000.

[14] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the ACM Conference on Knowledge Discovery and Data Mining*, pages 133–142, 2002.

[15] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Science*, 2007.

[16] O.Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *Proceeding of the ACM Conference on Information and Knowledge Management*, 2009.

[17] M. Taylor, J. Guiver, S. Robertson, and T. Minka. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the International Conference on Web Search and Web Data Mining*, pages 77–86, 2008.

[18] M. N. Volkovs and R. S. Zemel. Boltzrank: Learning to maximize expected ranking gain. In *Proceedings of the International Conference on Machine Learning*, pages 1089–1096, 2009.

[19] Q. Wu, C. Burges, K. M. Svore, and J. Gao. Adapting Boosting for Information Retrieval Measures. *Information Retrieval*, 2009.