Learning to Efficiently Rank

Lidan Wang
Dept. of Computer Science
University of Maryland
College Park, MD
lidan@cs.umd.edu

Jimmy Lin
The iSchool
University of Maryland
College Park, MD
jimmylin@umd.edu

Donald Metzler
3333 Empire Ave.
Yahoo! Research
Burbank, CA
metzler@yahoo-inc.com

ABSTRACT

It has been shown that learning to rank approaches are capable of learning highly effective ranking functions. However, these approaches have mostly ignored the important issue of efficiency. Given that both efficiency and effectiveness are important for real search engines, models that are optimized for effectiveness may not meet the strict efficiency requirements necessary to deploy in a production environment. In this work, we present a unified framework for jointly optimizing effectiveness *and* efficiency. We propose new metrics that capture the tradeoff between these two competing forces and devise a strategy for automatically learning models that directly optimize the tradeoff metrics. Experiments indicate that models learned in this way provide a good balance between retrieval effectiveness and efficiency. With specific loss functions, learned models converge to familiar existing ones, which demonstrates the generality of our framework. Finally, we show that our approach naturally leads to a reduction in the variance of query execution times, which is important for query load balancing and user satisfaction.

Categories and Subject Descriptors: H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval

General Terms: Algorithms, Performance

Keywords: learning to rank, linear models, effectiveness and efficiency tradeoff

1. INTRODUCTION

Web search engines allow users to find information on almost any topic imaginable. To be successful, a search engine must return the most relevant information to the user in a short amount of time. However, efficiency (speed) and effectiveness (relevance) are competing forces that often counteract each other. It is often the case that methods developed for improving relevance incur moderate-to-large computational costs. Therefore, sustained relevance gains must often be counter-balanced by buying more (or faster) hardware, implementing caching strategies if possible, or spending additional effort in low-level optimizations.

It is common for search engines to select a single operating point in the space of all possible efficiency-effectiveness tradeoffs. However, users and information needs are diverse. While most users may want their search results immediately, others may not mind waiting a little extra time if it means their results, on average, would be better. This same idea can be applied to information needs. Certain classes of queries, such as those for simple information needs, are expected to be answered immediately. However, for very complex information needs, users may be willing to incur additional latency for better results. Hence, operating at a "one size fits all" point along the tradeoff curve may not be optimal for all users and queries.

In this paper, we explore issues related to the efficiency-effectiveness tradeoff in the context of developing highly effective, highly efficient search engine ranking functions. We propose a framework for automatically learning ranking functions that optimize the tradeoff between efficiency and effectiveness. Traditional learning to rank approaches [15], have focused entirely on effectiveness. Therefore, it is more appropriate to think of our proposed approach as a learning to *efficiently* rank method. We will show that learning to rank (i.e., only optimizing effectiveness) is simply a special case of our proposed framework.

Our framework consists of two components. The first is a set of novel metrics for quantifying the tradeoff between efficiency and effectiveness. The second is an approach to optimizing the metrics for a class of linear ranking functions. As we will show, the framework is robust and effective. It can be used to learn a "one size fits all" ranking function, or be used to learn different ranking functions for different classes of users and information needs that may have their own unique efficiency-effectiveness tradeoff requirements.

There are four primary contributions of our work. First, we motivate and formulate the problem of jointly optimizing the effectiveness and efficiency of search engine ranking components. Second, we introduce novel metrics for quantifying the tradeoff between effectiveness and efficiency. We demonstrate that such metrics can robustly support a full spectrum of effectiveness-efficiency tradeoff scenarios that can easily be adapted to a variety of search tasks, or targeted towards specific user populations, depending on preferences. Third, we present learning to rank methods for optimizing our proposed tradeoff metrics. Experiments show that models learned in this way outperform traditional machine-learned models in terms of achieving an optimal

SIGIR'10, July 19–23, 2010, Geneva, Switzerland.

balance between effectiveness and efficiency. Finally, we discuss how our framework yields models with significantly reduced query execution time variance, which is a desirable property for load balancing and user satisfaction in practical search settings.

The remainder of this paper is laid out as follows: First, Section 2 surveys related work. Next, in Section 3 we propose novel metrics for quantifying the tradeoff between effectiveness and efficiency. Section 4 discusses methods for automatically learning models that are both effective and efficient by optimizing our proposed tradeoff metrics. Section 5 presents experimental results under different tradeoff scenarios. Finally, we conclude the paper and propose areas for future work in Section 6.

2. RELATED WORK

There has been a great deal of research devoted to developing efficient and effective retrieval systems. This has given rise to two distinct research threads. The focus of the first thread is on designing effective retrieval models. This has given rise to a steady stream of effectiveness-centric models, such as language models for information retrieval [20], the BM25 model [21], numerous term proximity models [16, 9, 23, 3], and learning to rank [12, 18, 6, 15]. The other thread is devoted to building efficient retrieval systems. Improved query execution strategies [22, 1] and advanced index pruning techniques [10, 8, 19] are just two examples of successful research directions along this thread.

The fact that these two threads are almost always investigated exclusively of each other has created a virtual dichotomy in the information retrieval research community. On one side there are researchers who develop highly effective, yet practically infeasible models and methods en masse. On the other side of the dichotomy are the researchers who design blazingly fast, yet spectacularly ineffective systems. One of the goals of our approach is to take a step towards eliminating this dichotomy by taking an efficiency-minded look at building effective retrieval models.

Our problem is quite different from previous work in index pruning [10, 7, 8, 1, 19] and query segmentation [4]. The primary goal of index pruning is to create a small index and search over this reduced index to gain better efficiency. In query segmentation, a syntactic parser is used to identify key term dependence features in a query, and only these key features, rather than all term dependence features, are used to retrieve documents. While both techniques are designed for dealing with query latency, these methods do not directly optimize the underlying efficiency and effectiveness metrics, e.g., optimizing index pruning or optimizing segmentation accuracy is not guaranteed to optimize retrieval effectiveness and efficiency, and their tradeoff.

Another way to speed up query evaluation is through caching [2] (i.e., term posting lists or query search results caching). Our problem and caching can be viewed as two complementary approaches for improving efficiency in search. Cached postings/results for a given query can be used during the query evaluation stage to improve efficiency, where the ranking function has been specified. In contrast, we learn efficient ranking functions, where a query evaluation strategy and a caching strategy are assumed to be given.

There have been several solutions proposed for dealing with the efficiency-effectiveness tradeoff in various contexts. First, in the machine learning community, it was shown that l_1 regularization is useful for "encouraging" models to have only a few non-zero parameters, thereby greatly decreasing the time necessary to process test instances [24]. Thus, l_1 regularized loss functions balance between model effectiveness (e.g., mean squared error, classification accuracy, etc.) and efficiency (number of non-zero parameters). However, quantifying efficiency in this way is overly simple and not very flexible. Indeed, the efficiency of most ranking functions can not be modeled simply as a function of the number of non-zero parameters, since the costs associated with evaluating different features are unlikely to be uniform (e.g., unigram scoring vs. term proximity scoring). The efficiency of a system ultimately depends on the specific implementation, architecture, etc. Therefore, l_1 regularization is too simple to be effective for jointly optimizing the effectiveness and efficiency of ranking functions.

In a similar direction, Collins-Thompson and Callan [11] investigated strategies for robust query expansion by modeling expansion term selection and weighting using convex programming. Their model included a variant of l_1 regularization that imposed a penalty for including common terms in the expanded query, since such terms would likely increase query execution time. This was the first effort that we are aware of that modeled efficiency in a search engine-specific manner. However, we note that query expansion and constructing ranking functions are two different problems and hence present different challenges. Furthermore, in this work, we model system efficiency using actual query execution times, instead of simple surrogates, such as term frequency, that may or may not accurately model the actual efficiency of the underlying retrieval system.

Finally, our work can be viewed as an enhancement of existing learning-to-rank strategies, which have, until now, focused exclusively on effectiveness. We expect the exploitation of efficiency in constructing effective ranking functions will allow for rapid development of highly effective *and* efficient retrieval models.

3. TRADEOFF METRICS

In this section we propose a novel class of *tradeoff metrics* that consider both effectiveness and efficiency. We begin by defining a general class of efficiency functions and describing how different functions yield different tradeoffs.

3.1 Measuring Efficiency

There are many different ways to measure the efficiency of a search engine, such as query execution time and queries executed per second. We are primarily interested in measuring how efficient a ranking function is at producing a ranked list for a *single query*. Throughout the remainder of this paper, we will assume that our measure of interest is query execution time, although any other query-level measure of efficiency could also be used.

Query execution times are, in theory, unbounded. This makes them difficult to work with from an optimization point of view. Instead, we would like to map query execution times into the range $[0, 1]$. We accomplish this by defining a function $\sigma(\cdot) : \mathbb{R}^+ \rightarrow [0, 1]$ that takes a query execution time, denoted by $\tau(Q)$ as input and returns an efficiency metric in the range $[0, 1]$, where 0 represents an inefficient ranking function and 1 represents an efficient ranking function.

We now define four different efficiency metrics. Each metric differs by how $\sigma(\cdot)$ is defined.

Constant. The most trivial efficiency metric is defined as $\sigma(Q) = c$, for $c \in [0, 1]$. This constant efficiency metric is always the same, regardless of the query execution time. This is the default assumption made by previous learning to rank approaches, which ignore efficiency altogether.

Exponential Decay. This loss function is defined as:

$$\sigma(Q) = \exp(\alpha \cdot \tau(Q))$$

where $\alpha < 0$ is a parameter that controls how rapidly the efficiency metric decreases as a function of query execution time. If a large (negative) decay rate (i.e., α) is specified, then the metric will drop off very quickly, penalizing all but the fastest query execution times.

Step Function. Often it is necessary to incorporate query execution time preferences into the efficiency metric. For instance, users may have a certain tolerance level for query execution time, such that they would expect the time to be less than a target t milliseconds for each query. A step function can naturally account for this requirement, as follows:

$$\sigma(Q) = \begin{cases} 1, & \text{if } \tau(Q) \leq t \\ 0, & \text{if } \tau(Q) > t \end{cases}$$

The step function metric is maximal (1) when query execution time is less than t and minimal (0) otherwise.

Step + Exponential Decay. If query execution time exceeds the threshold t in the step function efficiency metric, but only by a small amount, the metric assigned will still be 0, which may be overly harsh. Instead, it may be more reasonable to define a soft loss-like function, as follows:

$$\sigma(Q) = \begin{cases} 1, & \text{if } \tau(Q) \leq t \\ \exp(\alpha \cdot (\tau(Q) - t)), & \text{if } \tau(Q) > t \end{cases}$$

where $\alpha < 0$. The resulting function is a step function up until the threshold t and an exponential decay after time t with parameter α.

Figure 1 summarizes the four efficiency metrics just described. Note that there are many other ways to define $\sigma(\cdot)$ beyond those explored here. The best functional form for a given task will depend on many factors, including dataset size, hardware configuration, among others.

3.2 Measuring Effectiveness

There has been a great deal of research into evaluating the effectiveness of information retrieval systems. Therefore, we simply make use of existing effectiveness measures here. We define the effectiveness of a query Q as $\gamma(Q)$.

As with the efficiency metrics, we are primarily interested in effectiveness measures with range $[0, 1]$. Most of the commonly-used effectiveness metrics satisfy this property, including precision, recall, average precision, and NDCG. In this paper we will exclusively focus on average precision as the effectiveness metric of interest, although any of the above metrics can be substituted in our framework without loss of generality.

3.3 Efficiency-Effectiveness Tradeoff Metric

Our goal is to automatically learn ranking models that achieve an optimal middle ground between effectiveness and efficiency. However, before we can learn such a well-balanced model, we must define a new metric that captures the tradeoff. Our metric, which we call Efficiency-Effectiveness Tradeoff (EET), is defined for a query Q as the weighted harmonic mean of efficiency $\sigma(Q)$ and effectiveness $\gamma(Q)$:

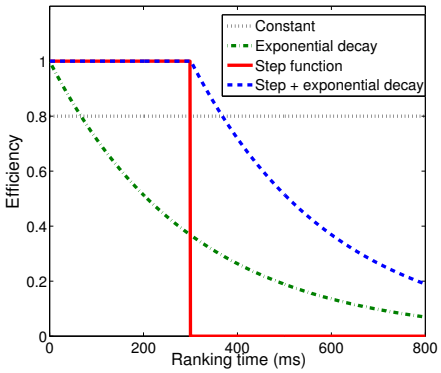

Figure 1: Efficiency functions. Constant and exponential decay are self explanatory. The step function and step + exponential function can model time preferences (threshold t=300ms here); when exceeding the time requirement, the ranking model either gets zero efficiency value or an exponentially lower efficiency value (respectively).

$$\text{EET}(Q) = \frac{(1 + \beta^2) \cdot (\gamma(Q) \cdot \sigma(Q))}{\beta^2 \cdot \sigma(Q) + \gamma(Q)}$$

where β is a parameter that controls the relative importance between effectiveness and efficiency. In this work, we set $\beta = 1$, which weighs both equally, but other settings can be trivially applied in our approach as well.

Given a ranking model R, the value of EET is computed for each query. To quantify the average tradeoff performance across N queries for a given ranking function, we define the following metric:

$$\text{MEET}(R) = \frac{1}{N} \sum \text{EET}(Q)$$

which is simply the mean EET value for the set of N queries.

It should now be clear that different choices of efficiency metrics will have a direct influence on MEET. For instance, a sharply decaying exponential efficiency metric represents a low tolerance for inefficient ranking models. Under such a function, the efficiency metric for a ranking function with high query execution time will likely be extremely low, resulting in a small MEET value, even if the ranking function is effective. On the other hand, if the efficiency function decays slowly or is constant, a ranking function with high effectiveness will also likely have a large MEET value.

Different combinations of efficiency metric and effectiveness metric will give rise to different MEET instantiations. Therefore, MEET is not a single metric, but a family of tradeoff metrics that depends on an efficiency component $\sigma(Q)$, an effectiveness component $\gamma(Q)$, and a tradeoff factor β.

4. LEARNING TO EFFICIENTLY RANK

In this section, we describe a class of linear ranking functions we will subsequently use to optimize our proposed tradeoff metrics. This simple class of ranking functions is used to show the benefits possible from optimizing MEET, rather than effectiveness alone.

Weighting	Description
$f_T(q, D) = \log\left[\frac{tf_{q,D}+\mu\frac{cf_q}{\|C\|}}{\|D\|+\mu}\right]$	Weight of unigram q in document D.
$f_O(q_j, q_{j+1}, D) = \log\left[\frac{tf_{\#1(q_j,q_{j+1}),D}+\mu\frac{cf_{\#1(q_j,q_{j+1})}}{\|C\|}}{\|D\|+\mu}\right]$	Weight of exact phrase "$q_j\ q_{j+1}$" in document D.
$f_U(q_j, q_{j+1}, D) = \log\left[\frac{tf_{\#uw8(q_j,q_{j+1}),D}+\mu\frac{cf_{\#uw8(q_j,q_{j+1})}}{\|C\|}}{\|D\|+\mu}\right]$	Weight of unordered window $q_j\ q_{j+1}$ (span = 8) in document D.

Table 1: Features used in the WSD ranking function. Here, $tf_{e,D}$ is the number of times concept e matches in document D, $cf_{e,D}$ is the number of times concept e matches in the entire collection, $|D|$ is the length of document D, and $|C|$ is the total length of the collection. Finally, μ is a weighting function hyperparameter.

4.1 Model

We focus our attention on a class of linear feature-based ranking functions that have the following form:

$$S(Q, D) = \sum_j \lambda_j f_j(Q, D) \quad (1)$$

where Q is a query, D is a document, $f_j(Q, D)$ is a feature function, and λ_j is the weight assigned to feature j. This ranking function is linear with respect to the model parameters λ. Various learning to rank approaches exist for finding parameters that optimize retrieval effectiveness metrics for these types of ranking functions [15]. One of the main benefits of such models is their ability to combine many different kinds of features in a straightforward manner, as we demonstrate below.

However, since we are also interested in optimizing for efficiency, we would like a mechanism for altering the efficiency characteristics of the ranking function. The most straightforward way to accomplish this is to eliminate one or more features from the ranking function. A logical way of choosing features to eliminate are those with small weights, as is done with l_1 regularization. We adopt a slight variant of this approach, where we assume that each weight λ also takes on a parametric form, as follows:

$$\lambda_j(Q) = \sum_i w_i g_i(Q)$$

where $g_i(Q)$ is a meta-feature that takes Q as input (discussed a bit later), and w_i is the weight assigned to the meta-feature. Notice that the weights λ are now *query dependent*, which means they can adapt better to different query scenarios via the feature functions g_i. We will show shortly that allowing λ to depend on Q provides an intuitive way to prune features.

Plugging these query-dependent weights into our original linear ranking function (Equation 1) gives us the following ranking function:

$$S(Q, D) = \sum_i w_i \sum_j g_i(Q) f_j(Q, D)$$

which is still a linear ranking function, but now with respect to w_i, which are the global, query-independent free parameters that must be learned.

As a concrete example of a highly effective ranking function that takes on this functional form, we consider the *weighted sequential dependence* (WSD) ranking function that was recently proposed by Bendersky et al. [5]. The WSD ranking function is formulated as:

Feature	Description
$g_1^t(q)$	# times q occurs in the collection
$g_2^t(q)$	# documents q occurs in the collection
$g_3^t(q)$	# times q occurs in ClueWeb
$g_4^t(q)$	# times q occurs in a Wikipedia title
$g_5^t(q)$	1 (constant feature)
$g_1^b(q_j, q_{j+1})$	# times bigram occurs in the collection
$g_2^b(q_j, q_{j+1})$	# documents bigram occurs in the collection
$g_3^b(q_j, q_{j+1})$	# times bigram occurs in ClueWeb
$g_4^b(q_j, q_{j+1})$	# times bigram occurs in a Wikipedia title
$g_5^b(q_j, q_{j+1})$	1 (constant feature)

Table 2: Meta-features used in WSD ranking.

$$\begin{aligned} S(Q, D) \quad = \quad & \sum_i w_i^t \sum_{q \in Q} g_i^t(q) f_T(q, D) + \\ & \sum_i w_i^b \sum_{q_j, q_{j+1} \in Q} g_i^b(q_j, q_{j+1}) f_O(q_j, q_{j+1}, D) + \\ & \sum_i w_i^b \sum_{q_j, q_{j+1} \in Q} g_i^b(q_j, q_{j+1}) f_U(q_j, q_{j+1}, D) \end{aligned}$$

where the features f_T and g_i^t are defined over query unigrams, and f_O, f_U, and g_i^b are defined over query bigrams. We define these features in a similar way as to how they were defined by Bendersky et al. [5]. Table 1 shows how the features f_T, f_O, and f_U are defined, while Table 2 summarizes the meta-features g_i^t and g_i^b. Hence, this specific ranking function consists of three general components: 1) a unigram term score, 2) a bigram phrase score, and 3) a bigram proximity score.

While the effectiveness of the ranking function depends on the weights w, models of this form provide a natural mechanism for eliminating features in a query-dependent manner, thereby improving efficiency. We propose to drop features according to the magnitude of $\lambda_i(Q)$, the query-dependent weight assigned to feature i. If $\lambda_i(Q)$ is nearly zero, then feature i is unlikely to have a significant impact on the final ranking. Therefore, it should be safe to drop feature i from the ranking function, thereby increasing the efficiency of the model, with minimal impact on effectiveness. This suggests the general strategy of pruning features if $|\lambda_i(Q)| \leq \epsilon$, where ϵ is a pruning threshold.

However, in this work, we are dealing with a model that we have specific domain knowledge about, and therefore use a more suitable pruning strategy. Previous work by Lease [13] demonstrated that unigrams have more positive impact on

retrieval effectiveness than bigrams; hence, we only prune bigram features from the WSD ranking function. Bigram features are pruned if they satisfy the following condition:

$$\frac{\lambda(q_i, q_{i+1})}{\lambda(q_i) + \lambda(q_{i+1})} \leq \epsilon$$

This condition says that if the ratio between the bigram feature weight and the sum of individual unigram feature weights is less than ϵ, then the bigram is eliminated. Preliminary experiments found the general strategy of pruning according to $|\lambda_i(Q)| \leq \epsilon$ to be effective, but found this ranking function-specific strategy to yield superior results. Therefore, it is likely that different ranking functions will require different pruning strategies to be maximally effective.

4.2 Parameter Estimation

We now describe our method for automatically learning the parameters of our proposed model from training data. We must not only learn the parameters w, but also the concept pruning threshold ϵ. Although there are many learning to rank approaches for learning a linear ranking function (i.e., estimating w), our optimization problem is complicated by the fact that we also have to learn the best ϵ, which is directly tied to the efficiency of the ranking function. Since the relationship between our metric and ϵ can not be modeled analytically, we are forced to directly estimate the parameters using a non-analytical optimization procedure.

We used a simple optimization procedure that directly optimizes MEET: a coordinate-level ascent algorithm similar to the one that was originally proposed in [17]. The algorithm performs coordinate ascent in the MEET metric space. Each (single dimensional) optimization problem is solved using a simple line search. Given that the MEET space is unlikely to be convex, there is no guarantee that this greedy hill climbing approach will find a global optimum, but, as we will show, it tends to reliably find good solutions for our particular problem. The final solution to the optimization problem is a setting of the parameters w and a pruning threshold ϵ that is a local maximum for the MEET metric.

This algorithm is used due to its simplicity and the fact that our model has a small number of parameters. Each function evaluation (i.e., MEET measurement) in the optimization procedure requires measuring both efficiency and effectiveness of the current parameter setting as applied to the training set. This can be costly for large training sets and large feature sets. There are several other approaches for directly optimizing non-smooth functions using similar types of hill-climbing methods, such as simultaneous perturbation stochastic approximation [25], which uses fewer function evaluations and could be used to speed up training. Although it is beyond the scope of our current work, we are very interested in exploring how to efficiently learn models with hundreds or even thousands of parameters.

5. EXPERIMENTS

In this section we present our experimental results. We begin by describing our experimental setup and then present a comprehensive evaluation of our proposed method using publicly available datasets.

5.1 Experimental setup

We implemented our learning to efficiently rank framework on top of Ivory, a newly-developed open-source web-scale information retrieval engine [14]. Experiments were run on a SunFire X4100, with two Dual Core AMD Opteron Processor 285 at 2.6GHz and 16GB RAM (although all experiments used only a single thread).

Name	Number of Docs	TREC Topics
Wt10g	1,692,096	451-550
Gov2	25,205,179	701-850
Clue	50,220,423	1-50

Table 3: Summary of TREC collections and topics used in our experiments.

To illustrate the benefits of our proposed work across a diverse set of document collections, we used the three TREC web collections shown in Table 3. Wt10g is a small web collection with 1.7 million documents, while Gov2 is a larger 25 million page crawl of the .gov domain. Finally, Clue is the first English segment of ClueWeb09, a recently-released web crawl consisting of 50 million documents.

We used the *title* portions of TREC topics as queries for these collections. In each case, queries were split sequentially into a training and test set of equal size; results are reported on the test sets. In all cases we ran retrieval on a single, monolithic index (i.e., no document partitioning), returning 1000 hits. Although real-world systems are likely to divide large document collections into smaller, independently-indexed partitions and coordinate parallel query evaluation via a broker, we decided not to adopt this strategy since it would conflate characteristics of the framework we wish to illustrate with unrelated issues such as latencies associated with network traffic. As a result, our query execution times on the larger collections may not be adequate for interactive retrieval; this, however, does not detract from the generality of our proposed framework.

We compare our proposed model, which we call the efficient sequential dependence model (ESD), to two baseline models. One is the bag-of-words query likelihood model [20] (QL), with Dirichlet smoothing parameter $\mu = 1000$. The other is the less efficient, but more effective sequential dependence model [16] (SD). The SD model is a special case of the WSD and ESD models. The best practice implementation of the SD model uses the same features and functional form as the WSD model, but sets $w_5^t = 0.82$, $w_5^b = 0.09$. All of the other parameters are set to 0, yielding *query independent* λ_i weights. The SD model does not prune features, meaning that all features are evaluated for every query.

Effectiveness is measured in terms of mean average precision (MAP), although as previously noted a variety of other effectiveness metrics can be substituted. As for efficiency, we explored several different efficiency functions, and analyzed the resulting impacts on the tradeoff between efficiency and effectiveness (detailed below). When training our model, we directly optimized the MEET metric. A Wilcoxon signed rank test with $p < 0.05$ was used to determine the statistical significance of differences in the metrics.

5.2 Results

In this section we describe the performance of our model in terms of its ability to optimally balance effectiveness and efficiency. We show the impact of different efficiency functions on the learned models and also present an analysis of the distribution of query times, demonstrating reduced variance. Finally, we show that with specific efficiency functions, our

"Slow" Decay Rate

	Wt10g ($t = 150ms, \alpha = -0.05$)			Gov2 ($t = 5s, \alpha = -0.1$)			Clue ($t = 7s, \alpha = -0.01$)		
	Time(s)	MAP	MEET	Time(s)	MAP	MEET	Time(s)	MAP	MEET
QL	0.168	21.51	21.75	1.97	31.94	31.96	4.09	20.75	20.55
SD	0.401	22.43^{*} (+4.2)	22.12 (+1.7)	7.09	33.57^{*} (+5.1)	32.91 (+2.9)	13.46	21.68 (+4.5)	21.41 (+4.2)
ESD	0.235	$24.04^{*}_{\dagger}$ (+11.8/+7.2)	$23.03^{*}_{\dagger}$ (+5.8/+4.1)	6.42	$34.74^{*}_{\dagger}$ (+8.7/+3.5)	$33.94^{*}_{\dagger}$ (+6.2/+3.1)	11.18	22.34^{*} (+7.7/+3.0)	22.14 (+7.7/+3.4)

"Fast" Decay Rate

	Wt10g ($t = 150ms, \alpha = -0.45$)			Gov2 ($t = 5s, \alpha = -0.35$)			Clue ($t = 7s, \alpha = -0.1$)		
	Time(s)	MAP	MEET	Time(s)	MAP	MEET	Time(s)	MAP	MEET
QL	0.168	21.51	21.03	1.97	31.94	31.87	4.09	20.75	20.53
SD	0.401	22.43^{*} (+4.2)	19.63^{*} (−7.1)	7.09	33.57^{*} (+5.1)	31.77 (−0.3)	13.46	21.68 (+4.5)	21.26 (+3.5)
ESD	0.215	23.42^{*} (+8.9/+4.4)	$21.55^{*}_{\dagger}$ (+2.5/+9.8)	5.46	33.65^{*} (+5.4/+0.2)	32.58 (+2.2/+2.5)	8.55	21.24 (+2.4/−2.0)	21.08 (+2.7/−0.8)

Table 4: Comparison between models under step + exponential efficiency function (slow decay on top, fast decay on bottom); parameters t (time threshold) and α (decay rate) are shown in the column headings for each collection. Symbol * denotes significant difference with QL; † denotes significant difference with SD. Percentage improvement shown in parentheses: over QL for SD, and over QL/SD for ESD.

learned models converge to either baseline query-likelihood or the weighted sequential dependence model, thus illustrating the generality of our framework in subsuming ranking approaches that only take into account effectiveness.

5.2.1 Tradeoff between effectiveness and efficiency

Table 4 presents results of two sets of experiments using the step + exponential function, with what we subjectively characterize as "slow" decay and "fast" decay. The time threshold t (below which efficiency is one) was chosen to be roughly halfway between the QL and SD running times for each collection. Due to the differences in collection size, it is unlikely that a common decay rate (α) is appropriate for all collections. Therefore, we manually selected a separate decay rate for each collection. Both t and α are shown in the column headings of Table 4. The fast decay function penalizes low efficiency ranking functions more heavily, thus a highly-efficient ranking function with reasonable effectiveness is preferred over a less efficient function with potentially better effectiveness. With the slow decay function, effectiveness plays a greater role.

For both fast and slow decay, we compared our proposed model (ESD) with the query likelihood model (QL) and the sequential dependence model (SD) in terms of query evaluation time, MAP, and MEET. In both tables, percentage improvements for MAP and MEET are shown in parentheses: over QL for SD, and over QL/SD for ESD. Statistical significance is denoted by special symbols.

As expected, the mean query evaluation time for ESD is greater than that of QL, but less than that of SD for both sets of experiments. Furthermore, the mean query evaluation time for ESD is lower for the fast decay rate than for the slow decay rate, which suggests that our efficiency loss function is behaving as expected. In the learned models, ESD is 41.4%, 9.4%, and 16.9% faster than SD for the slow decay rate on Wt10g, Gov2, and Clue, respectively; ESD is 46.4%, 23.0%, and 36.5% faster than SD for the fast decay rate on the same three collections, respectively. Once again, this makes sense, since the fast decay rate penalizes inefficient ranking functions more heavily.

In terms of mean average precision, in five out of the six conditions, the ESD model was significantly better than baseline QL. In the one condition in which this was not the case (Clue with fast decay), SD was not significantly better than QL either. While the ESD model is much more efficient than the SD model, it is able to retain the *same* effectiveness as SD, and in some cases actually performs *significantly better* (in the case of Wt10g and Gov2 for slow decay rate). We believe that this result demonstrates the ability of our framework to select a more optimal operating point in the space of effectiveness-efficiency tradeoffs than previous approaches.

In terms of MEET, the ESD model outperforms QL and SD, although gains are not consistently significant. Interestingly, we note that SD has a lower MEET score than default QL in two out of three collections when the fast decay rate is used. This suggests that the default formulation of the sequential dependence model trades off a bit too much efficiency for effectiveness, at least based on our metrics.

Lastly, note that setting the time target t in the efficiency function implies that the ranking model will be penalized by an exponential decay in efficiency if its query ranking time exceeds t. This is a soft penalization factor, which contrasts with the more harsh step function where efficiency is assigned a zero value for time exceeding t. An implication of using this soft efficiency loss function is that for a ranking function with time $> t$, if it is *highly* effective for user queries, it may still have a reasonable tradeoff value, because essentially, its high effectiveness compensates for the loss in efficiency. This fact is also confirmed by results shown in Table 4, where the average query time of ESD is consistently greater than the time threshold t.

5.2.2 Analysis of query latency distribution

Another benefit of our proposed framework is that learned models exhibit low variance in query execution times. Fig-

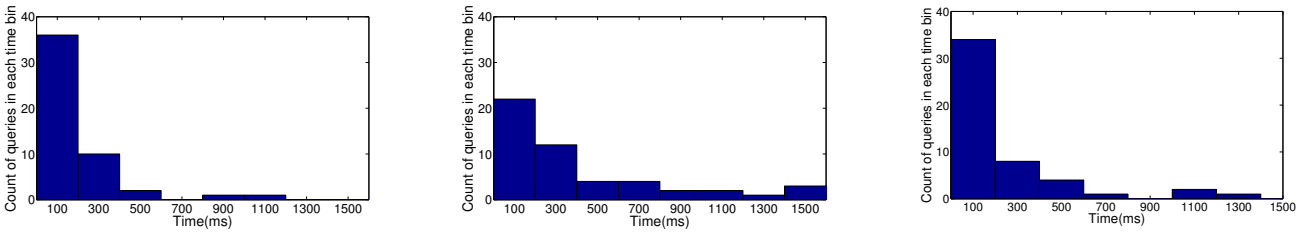

Figure 2: Distribution of query execution time for Wt10g queries for (i) query likelihood (QL); (ii) sequential dependence model (SD); (iii) efficient sequential dependence model (ESD).

ure 2 plots histograms of query execution for QL, SD, and ESD on the Wt10g collection. The ESD model was trained using the step + exponential (fast decay) efficiency function. Distributions for the other conditions look similar, and therefore we omit in the interest of space.

We can see that most queries with baseline QL are evaluated in a short amount of time, with a small number of outliers. The sequential dependence model has a heavier tail distribution with increased variance in query execution times: most queries still finish relatively quickly, but a significant minority of the queries take much longer to evaluate. Our ESD model reduces the number of long running queries so that the histogram is less tail heavy, which greatly improves the observed variance of query execution times. This improved behavior is due to the fact that our model considers efficiency as well as effectiveness, hence penalizes long-running queries, even if they are more effective. Note that although query likelihood has the most desirable query execution profile, it comes at the cost of effectiveness. Experiments in the previous section showed that ESD is at least as effective as SD, but much more efficient. The distribution of query execution times further supports this conclusion.

Why is reduced variance in query execution time important? For real-world search engines, it is important to ensure that a user, *on average*, gets good results quickly. However, it is equally important to ensure that *no user* waits too long, since these represent potentially dissatisfied users who never come back. A basic principle in human-computer interactions is that the user should never be surprised, and that system behavior falls in line with user expectations. Reducing the variance of query execution times helps us accomplish this goal.

Furthermore, from a systems engineering point of view, lower variance in query execution time improves load balancing across multiple servers. In real-world systems, high query throughput is achieved by replicated services across which load is distributed. If variance of query execution times is high, simple approaches (e.g., round-robin) can result in uneven loads (consider, for example, that in SD one query can take an order of magnitude longer than another to execute). Therefore, the reduced variance exhibited by our learned models is a desirable property.

5.2.3 Relationships to other retrieval models

Finally, we demonstrate that previous ranking models that consider effectiveness only can be viewed as special cases in our proposed family of ranking functions that account for both effectiveness and efficiency. More specifically, the flexible choices for efficiency functions used in our general framework can capture a wide range of tradeoff scenarios for different effectiveness/efficiency requirements. For instance, if we care more about efficiency than effectiveness, then we can set the time threshold in the efficiency function to be low, which forces us to learn a ranking function with high efficiency. On the other hand, if the focus is to learn the most effective ranking function possible (disregarding efficiency), then we can use a constant efficiency value. We would expect that in the first case, the learned model would look very similar to baseline query likelihood (efficient but not effective). Correspondingly, we would expect that in the latter case, the learned model would look very similar to the sequential dependence model (effective but not efficient). For particular choices of the efficiency function, the learned models should converge to (i.e., acquire similar parameter settings as) existing models that encode a specific effectiveness/efficiency tradeoff.

Table 5 compares ESD to the SD model with a constant efficiency function. Our model, when trained with constant efficiency values, is equivalent to the WSD model [5]. The ESD model in this case significantly outperforms SD in MAP and MEET scores; the differences are significant in two of the three collections.

Similarly, Table 6 illustrates the relationship of ESD to QL under a step efficiency function with low time targets (t = 100ms is used for Wt10g and t = 3s is used for Clue and Gov2). Step efficiency functions heavily penalize long query execution times, so the model essentially converges to simple bag of words. An interesting observation is that while retaining similar effectiveness as the QL model, the ESD model achieves a better time efficiency than QL due to its joint optimization of effectiveness and efficiency (allowing the model to prune query terms that have little impact on effectiveness, but nevertheless have an efficiency cost).

6. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a principled and unified framework for learning ranking functions that jointly optimize both retrieval effectiveness and efficiency. This new problem was motivated by the fact that although current learning to rank approaches can learn highly effective ranking functions, the important issue of efficiency has been ignored. For real-world search engines, both efficiency and effectiveness are important factors.

We proposed novel metrics to quantify the tradeoff between retrieval effectiveness and efficiency and presented a strategy for automatically learning models that directly optimize the tradeoff metrics. We demonstrated that these metrics can support a full spectrum of effectiveness-efficiency

	Wt10g			Gov2			Clue		
	Time	MAP	MEET	Time	MAP	MEET	Time	MAP	MEET
SD	0.401	22.43	22.44	7.09	33.57	33.27	13.46	21.68	21.42
ESD	0.425	$24.11_{\dagger}$ (+7.5)	$23.34_{\dagger}$ (+4.0)	7.13	$34.35_{\dagger}$ (+2.3)	$34.08_{\dagger}$ (+2.4)	13.87	22.43 (+3.5)	22.26 (+3.9)

Table 5: Comparison of SD and ESD under constant efficiency (i.e., only effectiveness is accounted for in the tradeoff metric).

	Wt10g			Gov2			Clue		
	Time	MAP	MEET	Time	MAP	MEET	Time	MAP	MEET
QL	0.168	21.51	13.37	1.97	31.94	25.82	4.08	20.75	16.02
ESD	0.145	21.50 (–)	13.50 (+1.0)	1.93	31.63 (−1.0)	25.39 (−1.7)	3.68	20.70 (−0.2)	16.02 (–)

Table 6: Comparison of QL and ESD under step efficiency functions. A step function with $t = 3$s is used for Clue and Gov2, and a step function with $t = 100$ms is used for Wt10g.

tradeoff scenarios. Furthermore, we described a class of linear feature-based models that can be directly optimized for our proposed tradeoff metrics.

Our experimental evaluation showed that a linear ranking function optimized in this manner is capable of balancing between retrieval effectiveness and efficiency, as desired. Our learned ranking functions achieved similar strong effectiveness as a state-of-the-art learning to rank model, but with significantly decreased average query execution times; the variance of query execution times was reduced as well. Therefore, our proposed framework is capable of learning highly effective models that are also efficient, which makes them desirable from a practical point of view.

There are several possible directions for future work. First, it would be interesting to apply our learning to efficiently rank framework to other ranking functions that use different types of features. Second, we would like to study scalable parameter estimation techniques that would allow us to train on ranking functions with hundreds or even thousands of features. Finally, we have only begun to scratch the surface in various efficiency functions that model different tradeoff scenarios. Due to limited space, we were only able to show a few possibilities under a small set of parameter settings. A more thorough exploration of the parameter space would help us better understand how to cater efficiency functions to real-world situations.

7. ACKNOWLEDGMENTS

This work was supported in part by the NSF under awards IIS-0836560 and IIS-0916043; by DARPA contract HR0011-06-02-001 (GALE). Any opinions, findings, conclusions, or recommendations expressed are the authors' and do not necessarily reflect those of the sponsors. The second author is grateful to Esther and Kiri for their loving support.

8. REFERENCES

[1] V. Anh and A. Moffat. Pruned query evaluation using pre-computed impacts. *SIGIR*, p. 372–379, 2006.

[2] R. Baeza-Yates, A. Gionis, F. Junqueira, V. Murdock, V. Plachouras, and F. Silvestri. The impact of caching on search engines. *SIGIR*, p. 183–190, 2007.

[3] J. Bai, Y. Chang, H. Cui, Z. Zheng, G. Sun, and X. Li. Investigation of partial query proximity in web search. *WWW*, p. 1183–1184, 2008.

[4] M. Bendersky, W. Croft, and D. Smith. Two-stage query segmentation for information retrieval. *SIGIR*, p. 810–811, 2009.

[5] M. Bendersky, D. Metzler, and W. Croft. Learning concept importance using a weighted dependence model. *WSDM*, p. 31–40, 2010.

[6] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. *ICML*, p. 89–96, 2005.

[7] S. Buttcher and C. Clarke. Efficiency vs. effectiveness in terabyte-scale information retrieval. *TREC*, 2005.

[8] S. Buttcher, C. Clarke, and P. Yeung. Indexing pruning and result reranking: Effects on ad-hoc retrieval and named page finding. *TREC*, 2006.

[9] S. Büttcher, C. Clarke, and B. Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. *SIGIR*, p. 621–622, 2006.

[10] D. Carmel, D. Cohen, R. Fagin, E. Farchi, M. Herscovici, Y. Maarek, and A. Soffer. Static indexing pruning for information retrieval systems. *SIGIR*, p. 43–50, 2001.

[11] K. Collins-Thompson and J. Callan. Query expansion using random walk models. *CIKM*, p. 704–711, 2005.

[12] F. Gey. Inferring probability of relevance using the method of logistic regression. *SIGIR*, p. 222–231, 1994.

[13] M. Lease. An improved Markov Random Field model for supporting verbose queries. *SIGIR*, p. 476–483, 2009.

[14] J. Lin, D. Metzler, T. Elsayed, and L. Wang. Of Ivory and Smurfs: Loxodontan MapReduce experiments for web search. *TREC*, 2009.

[15] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[16] D. Metzler and W. Croft. A Markov Random Field model for term dependencies. *SIGIR*, p. 472–479, 2005.

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

[18] R. Nallapati. Discriminative models for information retrieval. *SIGIR*, p. 64–71, 2004.

[19] A. Ntoulas and J. Cho. Pruning policies for two-tiered inverted index with correctness guarantee. *SIGIR*, p. 191–198, 2007.

[20] J. Ponte and W. Croft. A language modeling approach to information retrieval. *SIGIR*, p. 275–281, 1998.

[21] S. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. *TREC*, p. 109–126, 1994.

[22] T. Strohman, H. Turtle, and W. Croft. Optimization strategies for complex queries. *SIGIR*, p. 219–225, 2005.

[23] T. Tao and C. Zhai. An exploration of proximity measures in information retrieval. *SIGIR*, p. 295–302, 2007.

[24] R. Tibshirani. Regression shrinkage and selection via the lasso. *J. Roy. Statist. Soc. Ser. B*, 58(1):267–288, 1996.

[25] Y. Yue and C. Burges. On using simultaneous perturbation stochastic approximation for IR measures, and the empirical optimality of LambdaRank. *NIPS Machine Learning for Web Search Workshop*, 2007.