A Cascade Ranking Model for Efficient Ranked Retrieval

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
Information Sciences Institute
Univ. of Southern California
Marina del Rey, CA
metzler@isi.edu

ABSTRACT

There is a fundamental tradeoff between effectiveness and efficiency when designing retrieval models for large-scale document collections. Effectiveness tends to derive from sophisticated ranking functions, such as those constructed using learning to rank, while efficiency gains tend to arise from improvements in query evaluation and caching strategies. Given their inherently disjoint nature, it is difficult to jointly optimize effectiveness and efficiency in end-to-end systems. To address this problem, we formulate and develop a novel cascade ranking model, which unlike previous approaches, can simultaneously improve both top k ranked effectiveness and retrieval efficiency. The model constructs a cascade of increasingly complex ranking functions that progressively prunes and refines the set of candidate documents to minimize retrieval latency and maximize result set quality. We present a novel boosting algorithm for learning such cascades to directly optimize the tradeoff between effectiveness and efficiency. Experimental results show that our cascades are faster *and* return higher quality results than comparable ranking models.

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

General Terms: Algorithms, Performance

Keywords: learning to rank, effectiveness, efficiency

1. INTRODUCTION

There is often a tension between effectiveness and efficiency when building information retrieval systems. To achieve greater effectiveness (i.e., to deliver higher quality results), system designers are driven towards complex ranking functions that may combine evidence from dozens, hundreds, or even thousands of relevance signals, typically using sophisticated machine learning techniques [16]. This frequently comes at a cost in efficiency (i.e., a slower system), since complex ranking functions are computationally expensive, thus requiring more resources to achieve the same level of service. On the other hand, efficiency can be enhanced through a variety of approaches such as index pruning, feature pruning, approximate query evaluation, and systems engineering. However, most of these approaches degrade effectiveness, typically in ways that are difficult to control.

With the goal of achieving a better balance between retrieval effectiveness and efficiency, recent work has explored approaches to ranking that exhibit good tradeoff characteristics [27, 28]. In general, they work by eliminating features that are costly to compute and not predicted to contribute much to the quality of the results. While efficiency-minded feature selection is a natural way to make existing models faster, such pruning may negatively affect retrieval effectiveness. This problem is exacerbated for very large collections under tight efficiency constraints. In this setting, only a small handful of cheap features can be used for ranking, which can result in poor retrieval effectiveness.

We introduce a novel *cascade ranking model* for efficient ranked retrieval. Unlike previous approaches, the cascade uses a sequence of increasingly complex ranking functions to progressively prune documents and refine the rank order of non-pruned documents. Thus, the cascade model views retrieval as a multi-stage progressive refinement problem, where each stage considers successively richer and more complex ranking models, but over successively smaller candidate document sets. The intuition is that although complex features are generally more time-consuming to compute, additional overhead is offset by the fact that fewer documents are examined. This type of ranking paradigm is well-suited for large document collections, because the number of relevant documents is very small compared to the collection size. Hence, the ability to quickly hone in on a small set of candidate documents, via the cascade, can yield higher quality results *and* faster query execution times.

To achieve a desired efficiency-effectiveness tradeoff, we describe a novel boosting algorithm, a generalization of AdaRank [30], that jointly learns the model structure (i.e., optimal sequence of ranking stages) and the set of documents to prune at each stage. Experiments show that our cascade model can simultaneously improve effectiveness and efficiency compared to non-cascade feature-based models.

This paper has three major contributions. The first is the cascade model itself. Although similar coarse-to-fine-grained models have been used in other disciplines, to our knowledge this is the first application of this principle for learning an efficient ranking model. Second, we introduce a novel boosting algorithm for learning ranking cascades,

SIGIR'11, July 24–28, 2011, Beijing, China.

complete with a theoretical analysis. Finally, we carry out an extensive evaluation of our proposed model on several web collections. Results show that the cascade model consistently outperforms current state-of-the-art models, both in terms of efficiency and effectiveness.

The remainder of the paper is organized as follows: Section 2 describes related work. Section 3 details our proposed cascade ranking model, and Section 4 presents how cascade models can be learned. Next, Section 5 describes experimental results. Finally, Section 6 concludes the paper and outlines possible directions for future work.

2. RELATED WORK

In recent years, we have witnessed the success of machine learning approaches to the document ranking problem, as a whole known as learning to rank (e.g., [6, 17, 9, 30, 16], just to name a few). For the most part, however, these approaches have focused exclusively on effectiveness, sometimes leading to ranking functions that deliver high quality results, but are unbearably slow. Recently, however, a thread of work has emerged, dubbed learning to *efficiently* rank, that adopts an efficiency-minded approach. For example, regularization is useful for "encouraging" models to have few non-zero parameters, thereby serving as a crude form of feature pruning [24, 11]. More recently, Wang et al. [27, 28], in two separate studies, proposed linear ranking models that explicitly account for feature costs: the first is able to discover a more optimal point in the tradeoff space between efficiency and effectiveness, while the second is able to perform feature pruning to meet an externally-imposed time constraint. With feature pruning, the retrieval engine still implements a single monolithic ranking function—and thus it remains necessary to compute complex features for many documents (which is especially problematic for large web collections). In addition, since the number of non-relevant documents is significantly larger than the number of relevant documents in web-scale collections [8], applying a monolithic ranking model (even if used with a fast query evaluation engine) may waste computations, because a large number of the documents examined are non-relevant. In contrast, the cascade model considers increasingly complex features/ranking models on progressively fewer candidate documents. This approach, as we will show in the experiments, results in ranking functions that are simultaneously effective and efficient.

Our work is complementary to query evaluation [5, 25, 23]. Query evaluation focuses on how to efficiently evaluate a *given* ranking model, where we aim to *learn* an efficient ranking model in the first place. Along a similar direction, Cambazoglu et al. [8] explored early-exit strategies for additive ensembles. Although this approach bears some superficial resemblance to our cascade model, it is in fact completely different. They focus on optimizing the query evaluation process *given* a particular additive ensemble—and not about learning the ensemble. In contrast, our proposed model is accompanied by a boosting algorithm for learning optimal cascades according to a tradeoff metric.

A complementary approach to fast query evaluation is index pruning [10, 20]. Since query evaluation time monotonically increases with length of postings lists, shorter postings lists translate into faster queries. While discarding postings help reduce query latency, it does not directly optimize the underlying effectiveness and efficiency tradeoffs.

Finally, a variety of system engineering strategies exist to increase search efficiency, especially in operational settings. Caching [2, 21] reduces query latency significantly. Partitioning the document collection reduces latency, while replication of services increases throughout [12, 3]. These strategies are not IR-specific, but represent general principles for building large-scale distributed systems. In particular, we are not concerned with the system engineering aspects of efficiency (caching, partitioning, and replication), since these techniques can be applied to the cascade model just as they can be applied to any retrieval algorithm. These system engineering aspects can be viewed as orthogonal to our learning framework; their effects are captured by our analytical model of query execution time, which simply serves as an input to our task of learning an efficient ranking model.

The idea of progressive refinement in the cascade model is related to the general class of coarse-to-fine models that have been successfully applied in computer vision and machine learning [26, 29]. Most notably, Viola and Jones [26] tackled the problem of real-time face detection in images by a sequence of binary classifiers of increasing complexity, such that the most unlikely images are rejected early by simple classifiers, and the more promising object-like regions are passed to more complex classifiers for further consideration. However, these models are for high recall/precision applications and cannot be used for ranked retrieval, since they can only filter, but not *rank* items. As we will show, our proposed cascade is specifically built for achieving high top k *ranked* effectiveness in a very efficient manner.

3. CASCADE MODEL

In web search, there are significantly more non-relevant documents than relevant documents and most users only browse the top few results. Applying a monolithic ranking model for each query, even if used in conjunction with fast query evaluation techniques (e.g., [5, 25, 23]), may not be very efficient because a large number of scored documents are likely to be non-relevant and/or will not appear in the top k. Our proposed cascade model leverages these facts to achieve high *top k ranked* effectiveness in a highly efficient manner by constructing ranking models from simple to complex, applying the simple ones first, and pruning documents at each subsequent stage so that more complex (and better) ranking models are computed over fewer documents.

The cascade model consists of an additive sequence of stages $\{S_1 ... S_T\}$, where each stage S_t is associated with a pruning function J_t and a local ranking function H_t. Each stage receives as input the set of ranked documents from the previous stage and applies two sequential operations: first, the pruning function J_t is used to remove a number of documents from the input set (thus reducing the amount of effort involved in document scoring); then, the score contribution of the local ranking function H_t is added to the candidate documents still under consideration (to improve top k quality of the remaining documents). The results are forwarded to the next cascade stage for further pruning and re-ranking.

By construction, the cascade is arranged so that the local ranking functions increase in cost (and thus, complexity). Early stages take advantage of "cheap" ranking models to rank documents; the pruning functions discard documents that are unlikely to appear in the final top k. As a result, each successive stage is presented with a smaller candidate set, which enables the cascade to exploit more complex

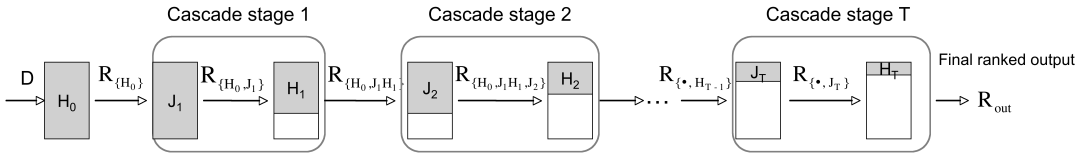


Figure 1: An example cascade. After an initial ranking function H_0, each stage consists of two sequential operations: J_t prunes the input ranked documents, then a local ranking function H_t refines the rank order of the retained documents. The new ranked list is passed to the next stage. The size of the shaded area denotes the size of the candidate documents. Subscripts for each ranked list denotes the sequence of actions applied.

and costly ranking models—hence improving effectiveness—without sacrificing efficiency.

As an example, Figure 1 presents a cascade model. The input to the cascade is a set of documents for a given query, and the final output is a ranked list of k documents (where k is specified in advance). An initial ranking function H_0 is applied to obtain an initial ranked list, $R_{\{H_0\}}$, which is then passed as input to the first stage. At the first cascade stage, the pruning function J_1 prunes documents in $R_{\{H_0\}}$ based on features of these documents (details in Section 3.1). The output from the pruning operation, denoted by $R_{\{H_0,J_1\}}$, is re-ranked by adding the contribution of H_1 to the document scores. This process repeats for the next cascade stage.

The overall score of a *non-pruned* document d_i at the end of a cascade model with T stages has the following form:

$$f_T(d_i) = \sum_{t=1}^{T} \alpha_t \cdot H_t \tag{1}$$

where α_t denotes the importance of the local model H_t. Following previous work in learning to rank, each H_t is a "weak ranker". We postpone discussing the actual ranking functions and our feature set until Section 4.

The iterative pruning and scoring mechanisms of the cascade provide a way to explicitly control the tradeoff between retrieval efficiency and ranked effectiveness. In terms of efficiency, the cascade aims to reduce the number of candidate documents at each stage:

$$|R_{\{\cdot,J_t\}}| \leq |R_{\{\cdot,J_{t-1}\}}| \leq \ldots |R_{\{\cdot,J_1\}}| \tag{2}$$

where each $|R_{\{\cdot,J_t\}}|$ denotes the resulting size of the documents after pruning at stage t, and the $\cdot$ abbreviates the previous sequence of pruning and re-ranking actions that have been applied to the input ranked documents R. In terms of effectiveness, the cascade aims to achieve the following:

$$E(R_{\{\cdot,H_t\}}) \geq E(R_{\{\cdot,H_{t-1}\}}) \geq \ldots E(R_{\{H_0\}}) \tag{3}$$

where $E(R_{\{\cdot,H_t\}})$ denotes the resulting top k effectiveness from applying H_t to refine the overall scores of the non-pruned documents that have reached S_t.

We can trivially obtain the most desirable outcome for either Equation (2) or Equation (3) at the expense of the other. If we set the pruning functions to never discard any documents, then the final ranked effectiveness $E(R_{\{\cdot,H_t\}})$ will be as high as possible since there will be no "loss" due to pruning. However, the cascade will likely be inefficient. Alternatively, if we prune every document, the result will almost certainly be fast, but ineffective. Thus, the objective is to design a well-balanced cascade by jointly learning the local ranking and pruning functions, guided by a tradeoff metric. We describe exactly such an algorithm in Section 4.

Before proceeding, a comment about the order in which pruning and re-ranking is performed at each stage: the pruning function is applied on the ranked documents produced from the *previous* stage, i.e., J_t reduces the size of the output documents from stage $t-1$. The reason behind this ordering is that while the local ranking function H_{t-1} used at the previous stage helps to refine the top k ranked effectiveness, pruning its re-ranked documents has a direct impact on the efficiency of stages $t, t+1, \ldots, T$. If the pruning is aggressive, then fewer documents will reach $t, t+1, \ldots, T$, thereby improving efficiency. Therefore, when learning the cascade, the pruning function defined for output documents from $t-1$ should (ideally) be jointly selected with the ranking functions at $t, t+1, \ldots, T$. For example, if H_t is complex, then pruning documents from $t-1$ must be aggressive to make it feasible to apply H_t; on the other hand, if H_t is simple, then more documents from $t-1$ may be kept and scored. While it would be ideal to jointly consider the pruning function with all subsequent ranking functions, this significantly complicates learning. Instead, we only consider the pruning function for documents produced from $t-1$ with the current ranking model H_t at stage t. To instantiate a cascade, we need to define the pruning functions, discussed next.

3.1 Pruning functions

At the input of each cascade stage t, we receive a set of ranked documents $R_{\{\cdot,H_{t-1}\}}$ passed from the previous stage, which are then filtered by the pruning function J_t. Since we have complete rank and score information for these input documents (up to stage $t-1$), the pruning function J_t can utilize their global rank and score. There are many ways to prune documents based on such global information: both rank-based [8] and score-based pruning methods [8, 1] have been proposed in the past. A key benefit of our model is that it is highly modular and flexible—the cascade is not restricted to a single pruning technique, but different stages can use different pruning functions J_t, which may be better suited to work with its corresponding local model H_t. This allows us to simply treat the different pruning methods as "pruning features", which can be selected at each stage. Our goal is not to develop novel pruning methods, but rather to use existing methods as building blocks within our model.

In this section, we present three pruning methods (J_t) that we have found to work well in our experiments. Each of these methods is parameterized by a pruning threshold β_t. The first two use document rank and score information to prune, while the third also considers the score distribution.

Rank-based. This pruning method uses document rank to eliminate a desired proportion of the input documents at each stage. The rank-based cutoff is defined as follows:

$$\textsc{RankCutoff}(\beta_t) : (1-\beta_t) \cdot |R_{\{\cdot,H_{t-1}\}}|$$

A document is pruned if it ranks below this cutoff value,

where β_t here is the pruning parameter, and $|R_{\{\cdot,H_{t-1}\}}|$ is the size of input documents at stage t. Large values of β_t lead to more aggressive pruning, i.e., $\beta_t = 1$ means all documents are discarded.

Score-based. Document scores provide another signal for pruning. Document scores for different queries are different, so enforcing a common score threshold is unlikely to work well. Instead, the score threshold is defined relative to the score range in each input document set:

$$\textsc{ScoreCutoff}(\beta_t) : \beta_t \cdot \textsc{ScoreRange}_{t-1} + \textsc{MinS}_{t-1}$$

Where $\textsc{ScoreRange}_{t-1}$ is defined to be the difference between the maximum and minimum scores in input $R_{\{\cdot,H_{t-1}\}}$ and $\textsc{MinS}_{t-1}$ is the minimum score in $R_{\{\cdot,H_{t-1}\}}$. This is equivalent to normalizing each score into [0, 1] by using the maximum and minimum scores in the candidate set. A document is pruned if it scores less than this threshold, where β_t is the pruning parameter. As before, large value of β_t leads to more aggressive pruning.

Mean-Max threshold. Often it is useful to consider the document score distribution for pruning. Several previous studies [14, 1] have considered the problem of inferring the score distributions of relevant and non-relevant documents, which are then used to help identify the best cutoff threshold for the top k documents in the ranked list to optimize a given evaluation metric. However, these methods only work for *set-based* measures such as F-measure and precision/recall, and do not work for top k ranked effectiveness measures.

We instead adopt a variant of this approach, and use a mean-max threshold function to capture characteristics of the score distribution, defined as a combination of the mean and the max of the input document scores:

$$\textsc{MeanMax}(\beta_t) : \beta_t \cdot \textsc{MaxS}_{t-1} + (1 - \beta_t) \cdot \textsc{MeanS}_{t-1}$$

Where $\textsc{MaxS}_{t-1}$ and $\textsc{MeanS}_{t-1}$ are the maximum and mean scores in input $R_{\{\cdot,H_{t-1}\}}$, respectively. A document is pruned if it scores less than this mean-max threshold. Similar approaches for using a mean-max threshold to control runtime scoring/prediction complexity have been used in the NLP task of structured prediction [29]. This formulation has the advantage that the pruning function can be better suited for each individual ranked list of documents.

4. LEARNING THE CASCADE

We now turn to the problem of learning a well-balanced cascade that optimizes a desired tradeoff between retrieval efficiency and ranked effectiveness. The entire cascade is defined by $\{<J_t(\beta_t), H_t, \alpha_t>\}$, for $t = 1, \ldots, T$: $J_t(\beta_t)$ is the pruning function and associated parameter (Section 3.1), and H_t is the local ranking model (described below) with its associated weight α_t.

Before we can learn a cascade, we must define how we measure top k ranked effectiveness and retrieval efficiency. For effectiveness, our primary measure is NDCG at k, although other metrics defined over top k rankings can easily be used instead. For retrieval efficiency, we use a cost model to estimate the execution cost of a given cascade. Retrieval engine details, such as query evaluation and caching strategies, are orthogonal to our general framework since their effects on query execution are captured by our cost model and simply serve as input to our learning algorithm.

4.1 Cost estimation

The total cost of cascade $\mathbf{S} = \{S_t\}$, $t = 1, \ldots, T$ for query q_i, denoted by $C(\mathbf{S}, q_i)$, is the sum of individual stage costs:

$$C(\mathbf{S}, q_i) = \sum_{t=1}^{T} C(S_t, q_i) \tag{4}$$

The cost of each stage is determined by the complexity of H_t and how many documents will be evaluated by H_t. We let U_t denote the unit cost of evaluating H_t over each document. The total cost of H_t at stage S_t is given by:

$$C(S_t, q_i) = U_t \cdot |R_{i\{\cdot,J_t\}}| \tag{5}$$

where $|R_{i\{\cdot,J_t\}}|$ denotes the size of the non-pruned documents after applying J_t. Intuitively, this cost model captures the fact that evaluating a more complex model over a large number of documents will result in greater time complexity.

The exact value of U_t depends on the implementation details of the search engine. Several previous studies have proposed strategies for estimating retrieval costs [7, 25]. The most common approach is directly fitting U_t to the actual query execution time of the ranking model [7]. We use this common approach for estimating U_t, where we run each H_t on the set of training queries, record its time, and set U_t to be the total time taken divided by the number of documents evaluated by H_t. For convenience, we normalize the unit costs so that the cheapest feature has a cost of one.

Finally, the query execution costs are unbounded, which makes them difficult to work with when learning a model. Therefore, we need to map the costs into the range $[0, 1]$. This is accomplished by using an exponential decay function $\exp(-\delta\ C(\mathbf{S}, q_i))$ to transform the cost into the $[0, 1]$ interval ($\delta = 0.01$ in our experiments). Other normalization techniques, such as computing the maximum cost (e.g., cost of applying the most expensive feature to every document in the collection) and then using it as a normalization factor, are also possible. However, this particular alternative may not differentiate costs very well since the cost distribution is likely to be skewed.

4.2 Tradeoff metric

The cascade learning problem is a multi-objective optimization problem [22]. The final objective metric is obtained by linearly combining the multiple objectives, which, in our case are the top k ranked effectiveness and the cost of the cascade model $\mathbf{S}$. For a given query q_i, the tradeoff is defined as follows:

$$\mathrm{T}(\mathbf{S}, q_i) = E(\mathbf{S}, q_i) - \gamma \cdot C(\mathbf{S}, q_i)$$

where $E(\mathbf{S}, q_i)$ represents ranked effectiveness, $C(\mathbf{S}, q_i)$ is the computational cost (Equation 4), and $\gamma \in [0, 1]$ is a parameter that controls the relative importance between effectiveness and efficiency. Note that $E(\mathbf{S}, q_i)$ and $E(f_T, q_i)$ mean the same thing, i.e., the effectiveness achieved by a cascade $\mathbf{S}$ with T stages (by Equation 1); in cases where we wish to draw attention to the ranking of the cascade, we use f_T for convenience. From the tradeoff definition, it should be clear that as we add more stages to a cascade, the total cost will increase. Therefore, in order to improve the tradeoff metric, the effectiveness gain from adding extra stages must counteract their costs.

Algorithm 1: Boosting algorithm for cascade learning

Initialize distribution $P_1(q_i) = 1/N$, where N is the number of queries and q_i denotes a query;

Initialize cascade model $\mathbf{S} = \{\}$;

for $t = 1, \ldots, T$ **do**

- Select a cascade stage $S_t = <J_t(\beta_t), H_t, \cdot>$ over the training instances weighted by P_t
- Set feature weight α_t for H_t: $\alpha_t = \frac{1}{2} ln \frac{\sum_{q_i} \frac{P_t(q_i)}{1-\gamma \cdot C(S_t,q_i)} \cdot (1+E(S_t,q_i))}{\sum_{q_i} \frac{P_t(q_i)}{1-\gamma \cdot C(S_t,q_i)} \cdot (1-E(S_t,q_i))}$;
- Add full stage $<J_t(\beta_t), H_t, \alpha_t>$ to $\mathbf{S}$;
- Update distribution P_{t+1}: $P_{t+1}(q_i) = \frac{exp(-E(\mathbf{S},q_i)) exp(\gamma \cdot C(\mathbf{S},q_i))}{\sum_{q_i} exp(-E(\mathbf{S},q_i)) exp(\gamma \cdot C(\mathbf{S},q_i))}$;

end

return cascade model $\mathbf{S}$

4.3 Learning

We now turn to the problem of learning the best cascade model. The general setup is that given a set of ranking features (described later in Section 4.4), pruning functions (Section 3.1), and training queries with their associated relevance judgments, we want to learn a cascade to optimize a given tradeoff metric, where the cascade model is characterized by $\{<J_t(\beta_t), H_t, \alpha_t>\}$, $t = 1, \ldots, T$.

We propose a novel boosting algorithm, a generalization of AdaRank [30], that jointly learns the cascade structure and parameters. It is important to note that we can not simply use AdaRank to optimize our tradeoff metric because AdaRank assumes linearity of its optimization metric O, i.e., $O(\mathbf{S}_{t-1} + \alpha_t S_t, q_i) \approx O(\mathbf{S}_{t-1}, q_i) + \alpha_t O(S_t, q_i)$, where $\mathbf{S}_{t-1}$ denotes the additive model up to stage $t-1$, S_t is a stage and α_t is the local weight [30, 16]. Note that each AdaRank stage consists of only H_t (a weak learner), since it does not perform document pruning. The tradeoff metric T does *not* satisfy the assumption because α_t in our case is not defined over the *entire* stage S_t, since in addition to H_t, the stage has a pruning function J_t for document reduction as well.

Our boosting algorithm for jointly optimizing top k ranked effectiveness *and* retrieval efficiency in a unified framework is shown in Algorithm 1. The algorithm proceeds in rounds to sequentially learn a set of cascade stages to optimize over weighted training instances. Each training instance (a query q_i) has an associated importance weight, denoted by $P_t(q_i)$. Initially, the weight distribution is set to uniform, and is updated at the end of each iteration. At each iteration, the parameterized pruning function $J_t(\beta_t)$ and the weak ranker H_t are first constructed based on the weighted training data. We describe this construction in more detail in Section 4.4.

Once $J_t(\beta_t)$ and H_t are chosen, the algorithm selects the local weight $\alpha_t > 0$ for the ranker H_t, where $E(S_t, q_i)$ and $C(S_t, q_i)$ in the formula denote the effectiveness and cost, respectively, from evaluating H_t on the *reduced* set of documents (after J_t). Intuitively, α_t captures the effectiveness of H_t over weighted training instances.

Once α_t is selected, we add the fully constructed stage to the current cascade model. The weight distribution P_{t+1} is then updated using the cost and effectiveness from the *overall* cascade (as defined in the previous section). The weights on the underperforming queries (i.e., queries that have poor ranked effectiveness, yet are expensive to compute) are increased, so the subsequent iteration can focus more on improving those hard queries. Note that H_0, the first stage in the cascade, is not associated with any pruning. Stage H_0 simply scores and passes a set of top hits $R_{\{H_0\}}$ to the first cascade stage (in our experiments, $|R_{\{H_0\}}| = 20,000$).

4.4 Cascade Stage Construction

In this paper, we use single features as weak rankers H_t, as in AdaRank [30]. Table 2 provides a summary of the features, which are similar to those in previous work (e.g., [18]). We use two families of scoring functions, based on the Dirichlet score from language modeling and BM25. Each family consists of a unigram feature, a bigram proximity feature that takes term order into account (parameterized with a window $S \in \{1, 2, 4\}$), and a bigram feature score for unordered terms (parameterized with a window $S' \in \{2, 4, 8\}$).

Typically, bigram features are computed over the entire query, which is problematic, as pointed out in Wang et al. [28]. Consider the query "white house rose garden": intuitively, the bigram "white house" is more important than "house rose". Computing features for *all* bigrams would be wasteful, so we need a mechanism to capture the importance of different query bigrams. It is accomplished by parameterizing bigram features with an "importance bin". Each query bigram occupies a bin, sorted by concept importance as measured by the *weighted sequential dependence* model [4]. Therefore, selecting the first bin amounts to selecting the most important query bigram. The cross of the feature and the bin is available to the learner to independently select from, thus allowing the cascade to selectively add query bigrams. Note that unigram features are *not* binned.

We note that our cascade model and learning algorithm can work with other ranking features beyond those defined here. Our approach can easily handle hundred or even thousands of features, the scale at which commercial search engines operate. The contribution of this work is not feature engineering, but rather the novel cascade ranking model and learning algorithm.

Let S_t denote the pair $< J_t(\beta_t), H_t >$, where J_t is a pruning function as defined in Section 3.1 and H_t is a weak ranker drawn from one of the features described above. At each boosting iteration, we select a stage S_t according to the following formula:

$$S_t = \arg\max_{S_t} \ \varphi_t^2 - \left[\sum_{q_i} \frac{P_t(q_i)}{1 - \gamma \cdot C(S_t, q_i)} \right]^2$$
$$\text{where } \varphi_t = \sum_{q_i} \frac{P_t(q_i)}{1 - \gamma \cdot C(S_t, q_i)} E(S_t, q_i) \qquad (6)$$

where $E(S_t, q_i)$ and $C(S_t, q_i)$ are effectiveness and cost, respectively, from computing H_t over the *reduced* set of documents (after pruning).[1] The goal of this optimization is to find the optimal combination of $J_t(\beta_t)$ and H_t that best balances cost and effectiveness over the weighted training data. Several methods can be used for this optimization, and in

[1]Note that for the purposes of the above arg max computation, α_t is irrelevant.

$$
\begin{aligned}
f_{T,Dir}(q,D) &= \log\left[\frac{\mathrm{tf}(q,D)+\frac{\mu}{|C|}\mathrm{cf}(q)}{|D|+\mu}\right] \\
f_{T,BM25}(q,D) &= \frac{(k_1+1)\cdot \mathrm{tf}(q,D)}{K+\mathrm{tf}(q,D)}\log\left[\frac{N-\mathrm{df}(q)+0.5}{\mathrm{df}(q)+0.5}\right] \\
f_{O,Dir,S}(q_j,q_{j+1},D) &= \log\left[\frac{\mathrm{tf}(\textsc{od}(S,q_j,q_{j+1}),D)+\frac{\mu}{|C|}\mathrm{cf}(\textsc{od}(S,q_j,q_{j+1}))}{|D|+\mu}\right] \\
f_{O,BM25,S}(q_j,q_{j+1},D) &= \frac{(k_1+1)\cdot \mathrm{tf}(\textsc{od}(S,q_j,q_{j+1}),D)}{K+\mathrm{tf}(\textsc{od}(S,q_j,q_{j+1}),D)}\log\left[\frac{N-\mathrm{df}(\textsc{od}(S,q_j,q_{j+1}))+0.5}{\mathrm{df}(\textsc{od}(S,q_j,q_{j+1}))+0.5}\right] \\
f_{U,Dir,S'}(q_j,q_{j+1},D) &= \log\left[\frac{\mathrm{tf}(\textsc{uw}(S',q_j,q_{j+1}),D)+\frac{\mu}{|C|}\mathrm{cf}(\textsc{uw}(S',q_j,q_{j+1}))}{|D|+\mu}\right] \\
f_{U,BM25,S'}(q_j,q_{j+1},D) &= \frac{(k_1+1)\cdot \mathrm{tf}(\textsc{uw}(S',q_j,q_{j+1}),D)}{K+\mathrm{tf}(\textsc{uw}(S',q_j,q_{j+1}),D)}\log\left[\frac{N-\mathrm{df}(\textsc{uw}(S',q_j,q_{j+1}))+0.5}{\mathrm{df}(\textsc{uw}(S',q_j,q_{j+1}))+0.5}\right]
\end{aligned}
$$

Figure 2: Definition of features used in our cascade model. $\mathrm{tf}(e,D)$ is the count of concept e in D, $\mathrm{df}(e)$ is the document frequency of concept e, $\mathrm{cf}(e)$ is the collection frequency of concept e, where e is defined as follows: q is a query term; $\textsc{od}(S,q_j,q_{j+1})$ is an ordered phrase, span of S ($S\in\{1,2,4\}$); $\textsc{uw}(S',q_j,q_{j+1})$ is an unordered phrase, span of S' ($S'\in\{2,4,8\}$). N is the number of documents in the collection; $|D|$ is the length of document D; $|D|'$ is the average document length in the collection; $|C|$ is the total length of the collection; for Dirichlet features, μ is a smoothing parameter; for BM25, $K=k_1[(1-b)-b\cdot(|D|/|D|')]$, and k_1, b are free parameters.

this work, we employ grid search [19] to find the set of H_t, J_t and β_t that maximizes the equation.

A formal analysis of the boosting algorithm is presented in Section 4.5 and the proof is provided in the Appendix. For now, we note that when the tradeoff parameter $\gamma=0$ (i.e., efficiency is ignored), the model simplifies to AdaRank. However, our algorithm can produce *both* effective and efficient ranking models and can be viewed as a generalization of AdaRank's effectiveness-only approach.

4.5 Analysis

In this section, we show how our boosting algorithm can continuously improve the tradeoff metric over the training data. We want to maximize the tradeoff metric T over the training queries:

$$\max_{\mathbf{S}} \sum_{q_i} \mathrm{T}(\mathbf{S},q_i) \tag{7}$$

which is equivalent to:

$$\min_{\mathbf{S}} \sum_{q_i} (1-\mathrm{T}(\mathbf{S},q_i)) \tag{8}$$

Because $1-x\le e^{-x}$ for any real value x, we minimize an exponential upper-bound of above expression:

$$\min_{\mathbf{S}} \sum_{q_i} exp(-\mathrm{T}(\mathbf{S},q_i)) \tag{9}$$

In our case, a linear combination of weak rankers is used to score the documents, with pruning performed at each stage. The optimization in Equation 9 is the same as:

$$\min_{S_t} \sum_{q_i} exp(-\mathrm{T}(\mathbf{S}_{t-1}\cup S_t,q_i)) \tag{10}$$

where $\mathbf{S}_{t-1}$ denotes the cascade up to stage $t-1$. For determining a single stage S_t, our boosting algorithm takes the approach of "forward stage-wise selection" [13], i.e., successively adding each cascade stage to improve the overall tradeoff metric. It can be proved that there exists a lower bound on the tradeoff metric over the training data:

$$T \ge 1-\prod_{t=1}^{T} e^{-\delta^t_{min}}\sqrt{\left[\sum_{q_i}\frac{P_t(q_i)}{1-\gamma\cdot C(S_t,q_i)}\right]^2-\varphi_t^2} \tag{11}$$

where φ_t is given in Equation 6, and let

$$\delta_i^t = E(f_{t-1}+\alpha_t H_t,q_i)-E(f_{t-1})-\alpha E(H_t,q_i)$$

where f_{t-1} denotes the linear combination of weak rankers up to $t-1$ (applied to the *non-pruned* documents only) and $\delta^t_{min}=\min_{i=1,\ldots,N}\delta_i^t$, where N denotes the number of queries. This means that the tradeoff metric can be continuously improved as long as the following holds:

$$e^{-\delta^t_{min}}\sqrt{\left[\sum_{q_i}\frac{P_t(q_i)}{1-\gamma\cdot C(S_t,q_i)}\right]^2-\varphi_t^2}<1 \tag{12}$$

That is, this condition is satisfied as long as the gain in effectiveness from additional stages is not outweighed by its cost. A detailed proof can be found in the Appendix.

5. EXPERIMENTS

This section presents experimental results: we first describe the experimental setup and implementation details, and then present an evaluation using TREC data.

5.1 Experiment Setup

Our cascade model was evaluated on three TREC web test collections: Wt10g, Gov2, and Clue (first English segment of ClueWeb09). Details for these collections are provided in Table 1. The topic titles were used as queries, split equally into a training and a test set. Model parameters were tuned on the training set; reported results are from the test set.

We compare our cascade model against a set of strong baselines in terms of top k ranked effectiveness and retrieval efficiency. Effectiveness is measured in terms of NDCG20 and precision at 20 (P20), while retrieval efficiency is measured in terms of average query execution time. Our cascade model is compared against three others: AdaRank [30],

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

which can be seen as a special case of the cascade model (i.e., optimized for effectiveness only with no efficiency considerations); a previously best-known model that jointly optimizes for both ranked effectiveness and efficiency by reducing the number of features computed at query time (which we call "FeaturePrune") [27]; and the basic query-likelihood model (QL). For fairness of comparison, "FeaturePrune" reimplements the approach and training method (greedy line search) described by Wang et al. [27], using the exact same feature set and objective function as our cascade model. As previously noted, since Cambazoglu et al. [8] focuses on early-exit strategies given a particular additive ensemble, it is difficult to meaningfully compare with our approach.

For training, we used NDCG20 as the effectiveness measure (E) in our tradeoff metric T, with γ set to 0.1. Both the cascade structure and the cascade parameters are automatically learned by directly optimizing the tradeoff metric over the training set. All results are reported over the test set. The Wilcoxon signed rank test with $p < 0.05$ was used to test for statistical significance.

Experiments were performed on a server running Red Hat Linux, with dual Intel Xeon quad-core processors (E5620 2.4GHz), 64GB RAM, and six 2TB 7.2K RPM SATA drives in RAID-6 configuration.

5.2 Implementation Details

All models were implemented in the Ivory open-source retrieval toolkit [15]. Baseline QL, AdaRank, and FeaturePrune work exactly as one might expect: by traversing postings in an inverted index and performing document-at-a-time scoring with max-score optimization [25]. The first stage of our cascade H_0 also works in the same way, using the weak learner that was selected by our boosting algorithm (retaining the top 20,000 hits). However, the remaining stages in the cascade adopt a different architecture. For stage H_1 and subsequent stages, we construct a forward index, which is essentially a list of pairs consisting of a document and a query term, grouped by the document. This structure can be efficiently built on the fly as we traverse postings in the initial cascade stage, by retaining the top documents as determined by the pruning function used in the first cascade stage. The forward index is small enough to be stored in memory and query evaluation for the subsequent stages is performed by iterating over the forward index. The reported retrieval efficiency of our cascade model accounts for the overall time taken by the cascade to return results, including the first stage. Other than this architectural difference, all models share exactly the same code, which makes for a fair comparison. Note that in all cases we used a single monolithic inverted index (i.e., no document partitioning). Based on the method described in Section 4.1, we computed U_T to be 1 for unigram and 20 for bigram features. This empirically matches actual retrieval times well.

One final implementation detail: to speed up pruning, our cascade allows pruning J_t to be performed "on-the-fly" within the computation of H_t, and so it incurs no additional cost. To see this, we observe that all three pruning methods prune input documents based on their rank order, i.e., a document with low score will be pruned before a document with high score. Thus, we simply iterate over the input documents in rank order, checking if each document d_i passes J_t, and if so, H_t is evaluated; else, the pruning/scoring process at stage t terminates (because if d_i does not pass pruning, any document ranked below it will not either). Note that descriptive statistics such as minimum, maximum, mean, etc. can be computed at the previous stage and passed to the pruning function. Coupling pruning J_t with H_t makes pruning extremely efficient.

5.3 Effectiveness vs. Efficiency

Table 2 reports NDCG20, P20, and average query evaluation time for our cascade model, QL, AdaRank, and the FeaturePrune method. For all three datasets, percentage improvements for both NDCG20 and P20 are shown in parentheses: over QL for AdaRank, and over QL/AdaRank for FeaturePrune and the cascade model. Statistical significance is denoted by special symbols in the table.

In all datasets, the cascade model achieves similar (and many times slightly better) effectiveness compared to AdaRank in both NDCG20 and P20, while being much faster. For instance, the cascade is 32.7%, 48.7%, and 34.7% faster than AdaRank on Wt10g, Gov2, and Clue, respectively. This means that our cascade model can equal or beat an effectiveness-only boosting model, while also being much faster. This illustrates that using a monolithic ranking function, as has been common practice for *ad hoc* retrieval, trades a great deal of efficiency for effectiveness. Such costly monolithic models are *not* more effective either since most of the documents they score are not relevant anyway. This also highlights the advantage of the cascade: by progressively reducing the size of candidate documents, it allows for the use of more complex ranking functions for high effectiveness without sacrificing efficiency.

Furthermore, we observe that the efficiency improvement of the cascade over AdaRank is greater for the two larger datasets (Gov2 and Clue) than Wt10g. This makes sense: compared to smaller document collections, larger collections contain more non-relevant documents. Thus, by filtering out these documents early in the ranking process, the cascade drastically improves efficiency and avoids evaluating documents that have little chance of appearing in the top k.

The cascade model also outperforms the feature pruning method in all evaluation measures across all datasets. In terms of retrieval time, the cascade is 12.9%, 44.5%, and 24.9% faster than the feature prune method on Wt10g, Gov2, and Clue, respectively. In terms of ranked effectiveness, the feature pruning method is slightly worse than AdaRank, likely due to removing ranking features for efficiency considerations. The FeaturePrune method behaves exactly as described in Wang et al. [27], discovering a better tradeoff point by giving up a bit of effectiveness for a gain in efficiency. However, the cascade model is able to obtain the best of both worlds: it can achieve better top k effectiveness *and* return results in a shorter amount of time.

Finally, compared to QL, which only uses simple term-based features for ranking and hence is very efficient, we observe that the cascade model is only slightly slower, but achieves much better top k effectiveness. Our cascade model

	Wt10g			Gov2			Clue		
	Time	NDCG20	P20	Time	NDCG20	P20	Time	NDCG20	P20
QL	0.080	34.07	32.40	1.15	44.57	50.93	2.60	27.50	34.20
AdaRank	0.260	35.49 (+4.2)	33.50 (+3.4)	3.90	47.37* (+6.3)	53.60 (+5.2)	6.55	**30.94*** (+12.5)	**37.40** (+9.4)
FeaturePrune	0.201	34.86 (+2.3/-1.8)	33.10 (+2.2/-1.2)	3.61	47.16 (+5.8/-0.7)	51.87 (+1.8/-2.1)	5.70	29.66 (+7.9/-4.1)	36.20 (+5.8/-3.2)
Cascade	0.175	**35.60** (+4.5/+0.3)	**33.80** (+4.3/+0.9)	2.00	**47.44*** (+6.4/0.1)	**54.47*** (+7.0/1.6)	4.28	30.60* (+11.3/-1.1)	**37.40** (+9.4/–)

Table 2: Comparison of retrieval time and effectiveness between query likelihood (QL), AdaRank, a feature-pruning method (FeaturePrune) and our cascade model. Effectiveness/efficiency tradeoff parameter γ is set to 0.1. Symbol * denotes sig. difference over QL. % improvement shown in parentheses: over QL for AdaRank, and over QL/AdaRank for FeaturePrune and Cascade. Time is measured in seconds.

	Wt10g			Gov2			Clue		
	NDCG20	Filtered	Filter loss	NDCG20	Filtered	Filter loss	NDCG20	Filtered	Filter loss
Stage 0	34.07	—	—	44.57	—	—	27.60	—	—
Stage 1	34.91	91.2%^{S}	0.09%	46.34	95.1%^{M}	0.15%	30.05	97.7%^{S}	0.09%
Stage 2	35.23	0.0%	0.0%	46.53	50.0%^{R}	1.6%	30.53	68.3%^{R}	0.18%
Stage 3	35.60	20.2%^{R}	0.04%	47.44	0.0%	0.0%	30.60	10.7%^{R}	0.0%

Table 3: For each stage of the cascade models in Table 2, we compute NDCG20, % documents filtered from the previous stage, and filter loss (% documents incorrectly pruned out of all documents passed from the *previous* stage). Values in the "Filtered" column are annotated with the pruning function that was learned: R for rank-based pruning, S for score-based pruning, and M for the mean-max threshold.

outperforms QL by 4.5%, 6.4% and 11.3% in NDCG20 on Wt10g, Gov2, and Clue, respectively, with the improvements on Gov2 and Clue statistically significant. Similar gains are also observed for P20.

5.4 Cascade Analysis

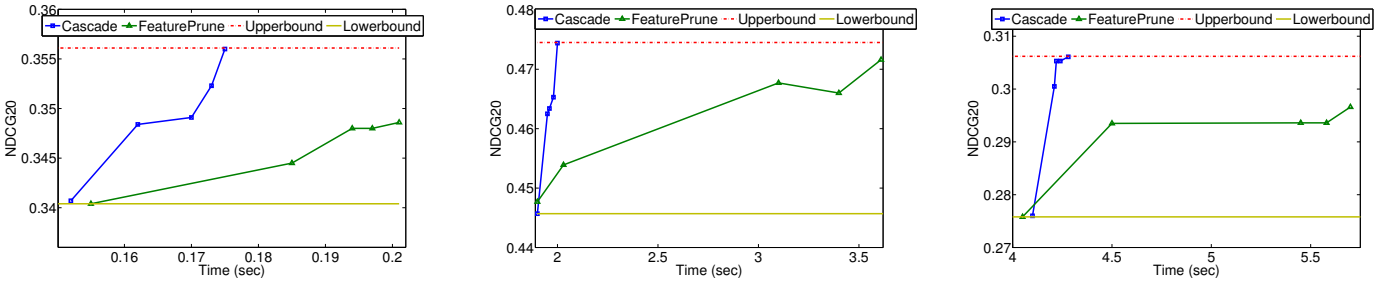
For the cascades learned in the previous section, we examine their behavior on a stage-by-stage basis in terms of effectiveness and efficiency. A detailed analysis is shown in Table 3: for each cascade stage, we present NDCG20 achieved up to that stage and the percentage of documents filtered from the *previous* stage. The values are annotated with the pruning function J learned by our boosting algorithm at each stage. We also compute filter loss, defined as the percentage of documents incorrectly filtered (i.e., relevant documents which are pruned) out of all documents passed from the *previous* stage. For all three collections, a tradeoff parameter of $\gamma = 0.1$ yields four stages. This is because the cost from adding additional stages outweighs the marginal gain in effectiveness, even with document pruning.

We see from Table 3 that in most cases, each cascade stage processes a substantially smaller set of documents than the previous stage, but always improves ranked effectiveness. As an example, by Stage 1, the cascade reduces the document set size by more than 90% in all three test collections, however, NDCG20 continues to improve in subsequent stages, due to using high quality/expensive ranking features over the small number of retained documents. For instance, our boosting algorithm learns to use simple term-based features in the initial stage for all three datasets, and uses term-proximity features (which are more costly) in subsequent stages to further improve the model's retrieval effectiveness. It is also interesting to see in all three datasets, the first stage prunes much more aggressively than subsequent stages. Because the input document set size is the largest at the first stage, the efficiency of the cascade can be significantly improved by eliminating a large number of documents early.

Also interesting is that in comparison to the NDCG20 achieved by FeaturePrune in Table 2 (which optimizes the same tradeoff), the cascade quickly achieves comparable effectiveness, and then surpasses it in subsequent stages. For instance, in comparison to Table 2, by stage 1, the cascade begins to surpass the final NDCG20 score achieved by FeaturePrune (Wt10g and Clue). By the final stage, the cascade NDCG20 scores surpass that achieved by AdaRank (Wt10g and Gov2). This illustrates that document pruning performed by the cascade improves efficiency while having minimal impact on effectiveness, compared to the monolithic ranking models. This is confirmed by the very low filter loss reported in the table (nearly zero for all stages).

We also observe that at stage 2 for Wt10g and stage 3 for Gov2, the cascade does not prune any input documents. This behavior can be explained by the tradeoff metric—the effectiveness gain from applying the ranking feature on all input documents outweighs its cost, and therefore the optimal pruning parameters at these stages are zero (i.e., no pruning). However, interestingly, the learned cascade for Clue, the largest collection, *always* prunes at all stages, and much more aggressively (e.g., at 97.7%, 68.3% and 10.7%) than the same stages for the two smaller collections. This is because web-scale collections contain a greater proportion of non-relevant documents; to combat this, the model learns that more aggressive pruning is necessary.

Finally, further analysis reveals that relevant documents that are filtered by our cascade are not ranked in the top k documents by AdaRank either, i.e., these documents have no chance of entering the top k even if an effectiveness-centric model is used. The cascade model is able to obtain the best of both worlds: it can achieve better top k effectiveness *and* return results in a shorter amount of time, compared to both AdaRank and the feature-pruning approach.



(i) Wt10g (ii) Gov2 (iii) Clue

Figure 3: NDCG20 as a function of time, generated by varying $\gamma \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

5.5 Parameter Variations

Our final set of experiments explores the effects of varying γ, the tradeoff parameter that balances effectiveness E and cost C in our objective function T. The setting of γ affects the cascade model and the feature pruning method, but not AdaRank (since it does not take into account efficiency) or the QL baseline (since no training is involved). Figure 3 shows NDCG20 of our cascade model and the feature-pruning method as function of average query evaluation time for each of the three collections, where each point represents a setting of γ, selected from the set $\{0.1, 0.3, 0.5, 0.7, 0.9\}$. Different values of γ produce different effectiveness/efficiency tradeoffs: a high value penalizes costly ranking functions, thus yielding faster models, whereas a smaller value yields more effective models. In each graph, the effectiveness lower bound, defined as the minimum effectiveness achieved under any condition, is plotted as the lower solid line, and the effectiveness upper bound, defined as the maximum effectiveness achieved, is plotted as the upper dotted line.

While in general effectiveness improves for both the cascade and the feature-pruning method when given more time for ranking, the cascade model consistently achieves equal or better NDCG20 across all conditions. It also approaches the upper bound more rapidly as time increases. Although both the cascade model and the feature-pruning method are able to realize different effectiveness/efficiency tradeoffs, these results show that the cascade model is superior in being able to return higher quality results much faster.

We also note that for Gov2 and Clue, the cascade tradeoff curve rises more steeply than for Wt10g. This bolsters our argument that the cascade model works particularly well for large collections. From the point of view of top k ranked effectiveness and efficiency, applying ranking features on a small number of documents is considerably more efficient but can also be more effective (by eliminating many non-relevant documents from consideration).

6. CONCLUSIONS AND FUTURE WORK

In this paper, we introduce a cascade ranking model for efficient top k retrieval, where a sequence of increasingly complex ranking functions is used to progressively refine a shrinking set of candidate documents. We propose a novel boosting-based algorithm that jointly learns the model structure (i.e., the optimal sequence of ranking stages) and the pruning criteria at each stage. Experiments show that our cascade model is able to *simultaneously* achieve high effectiveness and fast retrieval.

There are several future directions. We have only begun to explore the design space of machine learning algorithms for multi-objective optimization in the context of learning to rank: possibilities include other types of objectives (beyond linear combinations), other pruning functions (that take into account more than document score and rank), and different types of weak rankers (for example, decision trees). More work along these lines can further contribute to this emerging thread of learning to *efficiently* rank.

7. ACKNOWLEDGMENTS

This work has been supported by NSF under awards IIS-0836560, IIS-0916043, and CCF-1018625. 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 and dedicates this work to Joshua and Jacob.

8. REFERENCES

[1] A. Arampatzis, J. Kamps, and S. Robertson. Where to stop reading a ranked list? *SIGIR 2009*.

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

[3] L. Barroso and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan & Claypool, 2009.

[4] M. Bendersky, D. Metzler, and W. Croft. Learning concept importance using a weighted dependence model. *WSDM 2010*.

[5] E. Brown. Fast evaluation of structured queries for information retrieval. *SIGIR 1995*.

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

[7] F. Cacheda, V. Plachouras, and I. Ounis. A case study of distributed information retrieval architectures to index one terabyte of text. *IP&M*, 41:1141–1161, 2005.

[8] B. Cambazoglu, H. Zaragoza, O. Chapelle, J. Chen, C. Liao, Z. Zheng, and J. Degenhardt. Early exit optimizations for additive machine learned ranking systems. *WSDM 2010*.

[9] Y. Cao, J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon. Adapting ranking SVM to document retrieval. *SIGIR 2006*.

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

[11] K. Collins-Thompson. Reducing the risk of query expansion via robust constrained optimization. *CIKM 2009*.
[12] J. Hamilton. On designing and deploying Internet-scale services. *LISA 2007*.
[13] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2009.
[14] E. Kanoulas, V. Pavlu, K. Dai, and J. Aslam. Modeling the score distribution of relevant and non-relevant documents. *ICTIR 2009*.
[15] J. Lin, D. Metzler, T. Elsayed, and L. Wang. Of Ivory and Smurfs: Loxodontan MapReduce experiments for web search. *TREC 2009*.
[16] T.-Y. Liu. Learning to rank for information retrieval. *FnTIR*, 3(3), 2009.
[17] I. Matveeva, C. Burges, T. Burkard, A. Laucius, and L. Wong. High accuracy retrieval with multiple nested ranker. *SIGIR 2007*.
[18] D. Metzler. Automatic feature selection in the Markov random field model for information retrieval. *CIKM 2007*.
[19] J. Nocedal and S. Wright. *Numerical optimization*. Springer, 2006.
[20] A. Ntoulas and J. Cho. Pruning policies for two-tiered inverted index with correctness guarantee. *SIGIR 2007*.
[21] G. Skobeltsyn, F. Junqueira, V. Plachouras, and R. Baeza-Yates. ResIn: A combination of results caching and index pruning for high-performance web search engines. *SIGIR 2008*.
[22] R. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. John Wiley, 1986.
[23] T. Strohman, H. Turtle, and W. Croft. Optimization strategies for complex queries. *SIGIR 2005*.
[24] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society*, 58(1):267–288, 1996.
[25] H. Turtle and J. Flood. Query evaluation: Strategies and optimizations. *IP&M*, 31(6):831–850, 1995.
[26] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2002.
[27] L. Wang, J. Lin, and D. Metzler. Learning to efficiently rank. *SIGIR 2010*.
[28] L. Wang, D. Metzler, and J. Lin. Ranking under temporal constraints. *CIKM 2010*.
[29] D. Weiss and B. Taskar. Structured prediction cascades. *13th Conference on AI and Statistics*, 2010.
[30] J. Xu and H. Li. AdaRank: A boosting algorithm for information retrieval. *SIGIR 2007*.

Appendix

Here we prove Equation 11. For notational clarity, let $E(f_T, q_i)$ and $E(H_T, q_i)$ denote the effectiveness of cascade $\mathbf{S}_T$ and a stage S_T, respectively.

PROOF. Let: $Z_T = \sum_{q_i} exp(-E(f_T, q_i))exp(\gamma C(\mathbf{S}_T, q_i))$, $\phi_T = \frac{\sum_{q_i} \frac{P_T(q_i)}{1-\gamma C(S_T,q_i)} + \varphi_T}{2}$. Using definitions of $P_T(q_i)$, φ_T, and α_T in Section 4, we get $e^{\alpha_T} = \sqrt{\frac{2\phi_T}{2(\phi_T-\varphi_T)}} = \sqrt{\frac{\phi_T}{(\phi_T-\varphi_T)}}$.

$$
\begin{aligned}
Z_T = & \sum_{q_i} exp(-(E(f_T, q_i) - \gamma C(\mathbf{S}_T, q_i))) \\
= & \sum_{q_i} exp\{-E(f_{T-1}, q_i) - \alpha_T E(H_T, q_i) - \delta_i^T \\
& \qquad +\gamma C(\mathbf{S}_{T-1}, q_i) + \gamma C(S_T, q_i)\} \\
\leq & e^{-\delta_{min}^T} \sum_{q_i} exp(-E(f_{T-1}, q_i))exp(-\alpha_T E(H_T, q_i)) \\
& \qquad \cdot exp(\gamma C(\mathbf{S}_{T-1}, q_i))exp(\gamma C(S_T, q_i)) \\
= & e^{-\delta_{min}^T} Z_{T-1} \sum_{q_i} P_T(q_i) exp(-\alpha_T E(H_T, q_i)) \\
& \qquad \cdot exp(\gamma C(S_T, q_i))
\end{aligned}
$$

Since $e^x \leq \frac{1}{1-x}$ for any real x we have:

$$
Z_T \leq e^{-\delta_{min}^T} Z_{T-1} \sum_{q_i} \frac{P_T(q_i)}{1-\gamma C(S_T, q_i)} exp(-\alpha_T E(H_T, q_i))
$$

Since $E(H_T, q_i) \in [-1, 1]$ we have:

$$
\begin{aligned}
Z_T \leq & e^{-\delta_{min}^T} Z_{T-1} \sum_{q_i} \frac{P_T(q_i)}{1-\gamma C(S_T, q_i)} \{ \frac{1+E(H_T, q_i)}{2} e^{-\alpha_T} \\
& \qquad + \frac{1-E(H_T, q_i)}{2} e^{\alpha_T} \} \\
= & e^{-\delta_{min}^T} Z_{T-1} \left(\phi_T e^{-\alpha_T} + (\phi_T - \varphi_T) e^{\alpha_T} \right) \\
= & Z_1 \prod_{t=2}^{T} e^{-\delta_{min}^t} \sqrt{4\phi_t(\phi_t - \varphi_t)} \\
= & N \sum_{q_i} \frac{1}{N} exp(-E(f_1, q_i))exp(\gamma C(S_1, q_i)) \\
& \cdot \prod_{t=2}^{T} e^{-\delta_{min}^t} \sqrt{4\phi_t(\phi_t - \varphi_t)} \\
\leq & N\ e^{-\delta_{min}^1} \sum_{q_i} exp(-E(f_1, q_i))exp(\gamma C(S_1, q_i)) \\
& \qquad \cdot \prod_{t=2}^{T} e^{-\delta_{min}^t} \sqrt{4\phi_t(\phi_t - \varphi_t)} \\
\leq & N\ e^{-\delta_{min}^1} \sqrt{4\phi_1(\phi_1 - \varphi_1)} \prod_{t=2}^{T} e^{-\delta_{min}^t} \sqrt{4\phi_t(\phi_t - \varphi_t)} \\
= & N \prod_{t=1}^{T} e^{-\delta_{min}^t} \sqrt{4\phi_t(\phi_t - \varphi_t)}
\end{aligned}
$$

Substitute in ϕ_t and φ_t:

$$
\leq N \prod_{t=1}^{T} e^{-\delta_{min}^t} \sqrt{\left[\sum_i \frac{P_t(q_i)}{1-\gamma C(S_t, q_i)} \right]^2 - \varphi_t^2}
$$

So we have:

$$
\begin{aligned}
T = & \frac{1}{N} \sum_{q_i} E(f_T, q_i) - \gamma C(\mathbf{S}_T, q_i) \\
\geq & \frac{1}{N} \sum_{q_i} 1 - exp(1 - (E(f_T, q_i) - \gamma C(\mathbf{S}_T, q_i))) \\
\geq & 1 - \prod_{t=1}^{T} e^{-\delta_{min}^t} \sqrt{\left[\sum_i \frac{P_t(q_i)}{1-\gamma C(S_t, q_i)} \right]^2 - \varphi_t^2}
\end{aligned}
$$

A special case is when $\gamma = 0$ (effectiveness-only), this bound is exactly the same as AdaRank's. However, non-zero γ values will induce cascades of various tradeoff behaviors.