ABSTRACT

Title of dissertation: LEARNING TO EFFICIENTLY RANK

Lidan Wang, Doctor of Philosophy, 2012

Dissertation directed by: Associate Professor Jimmy Lin (primary advisor)
Professor Douglas Oard (co-advisor)
Department of Computer Science
and College of Information Studies

Web search engines allow users to find information on almost any topic imaginable. To be successful, a search engine must return 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 effectiveness incur moderate-to-large computational costs, thus sustained effectiveness gains typically have to be counter-balanced by buying more/faster hardware, implementing caching strategies if possible, or spending additional effort in low-level optimizations.

This thesis describes the "Learning to Efficiently Rank" framework for building highly effective ranking models for Web-scale data, without sacrificing run-time efficiency for returning results. It introduces new classes of ranking models that have the capability of being simultaneously fast and effective, and discusses the issue of how to optimize the models for speed and effectiveness. More specifically, a series of concrete instantiations of the general "Learning to Efficiently Rank"

framework are illustrated in detail. First, given a desired tradeoff between effectiveness/efficiency, efficient linear models, which have a mechanism to directly optimize the tradeoff metric and achieve an optimal balance between effectiveness/efficiency, are introduced. Second, temporally constrained models for returning the most effective ranked results possible under a time constraint are described. Third, a cascade ranking model for efficient top-K retrieval over Web-scale documents is proposed, where the ranking effectiveness and efficiency are simultaneously optimized. Finally, a constrained cascade for returning results within time constraints by simultaneously reducing document set size and unnecessary features is discussed in detail.

LEARNING TO EFFICIENTLY RANK

by

Lidan Wang

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2012

Advisory Committee:
Associate Professor Jimmy Lin, Chair
Professor Douglas Oard, Co-chair
Assistant Professor Hal Daume III
Associate Professor Amol Deshpande
Professor Carol Espy-Wilson
Dr. Donald Metzler

© Copyright by
Lidan Wang
2012

Dedication

To my parents.

// Acknowledgments

I have truly enjoyed my Ph.D. journey, and would like to thank a number of people who have made this thesis possible.

First and foremost, I would like to thank my advisor, Jimmy Lin, for his continuous help and support during my graduate career. I started working on this thesis topic several years ago, and during that initial period, if it were without Jimmy's support and belief in me, this thesis would have been entirely impossible, let alone see it to completion. But more importantly, Jimmy has played a pivotal role in my development as a researcher. From the early stage of my dissertation to the end, I have benefited tremendously from his insightful guidance and high standard of conducting original research. I am equally grateful to his trust in me to explore freely on my own when I found my own path.

I would like to thank Donald Metzler for his invaluable guidance and the many stimulating conversations we had on various topics in this thesis. From my collaborations with Don, I have learned how to think and ask the right research questions, and how to become a better technical writer – two of the most important qualities of a researcher. I feel very fortunate to have Don as a mentor.

Many thanks to Douglas Oard, who first introduced me to the field of information retrieval. He has been an excellent teacher who has helped me with my ideas with the breadth of his knowledge in this field.

I sincerely thank Hal Daume III for his advice and help during my job search process, and for his service on my thesis committee. I would also like to thank

Amol Deshpande for his guidance during my first year at Maryland, and Carol Espy-Wilson for her time to serve on my thesis committee.

Thanks are also due to Samir Khuller for his support during my Ph.D. study, and Paul N. Bennett, Kevyn Collins-Thompson, Susan Dumais, and Ryen White, for their guidance during my internship at Microsoft Research.

At the risk of forgetting someone, I would also like to thank my colleagues and friends in the CLIP lab and CS Department – Tamer Elsayed, Ferhan Ture, J. Scott Olsson, Bhargav Kanagal, Prithviraj Sen, Nick Frangiadakis, Jian Li, Zhongqiang Huang, Hossam Sharara, Walaa Moustafa, Grecia Lapizco-Encinas, Denis Filimonov, Galileo Namata, Mustafa Bilgic, Kristy Hollingshead Seitz, Hendra Setiawan, Ke Zhai, and Yuening Hu, for their invaluable friendship and discussions on many useful things.

Finally, I owe my deepest thanks to my parents - who have always stood by me, and offered encouragement when I needed it most. I dedicate my thesis to them.

Table of Contents

List of Figures viii

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Overview of learning to efficiently rank . . . 3
  - 1.2.1 Problem setup . . . 3
  - 1.2.2 Constructing efficient and effective linear models . . . 5
- 1.3 Contributions . . . 9
- 1.4 Outline of chapters . . . 11

2 Related Work and Background 12
- 2.1 Ranking models . . . 12
- 2.2 Query evaluation and caching strategies . . . 16
  - 2.2.1 Early exit strategies . . . 17
  - 2.2.2 Caching . . . 19
  - 2.2.3 Index pruning/query segmentation . . . 20
- 2.3 Coarse-to-fine models . . . 20
- 2.4 Other work that goes beyond effectiveness . . . 22

3 Efficient Linear Models 25
- 3.1 Background on Linear Feature-based Models . . . 27
  - 3.1.1 Linear feature-based models . . . 28
  - 3.1.2 SD and WSD models . . . 29
- 3.2 Basic setup . . . 33
- 3.3 Tradeoff metrics . . . 33
  - 3.3.1 Measuring efficiency . . . 34
  - 3.3.2 Measuring effectiveness . . . 37
  - 3.3.3 Efficiency-effectiveness tradeoff metric . . . 37
- 3.4 Efficient linear models . . . 40
  - 3.4.1 Limitations of linear models and offline feature selection . . . 40
  - 3.4.2 Efficient linear models . . . 42
- 3.5 Parameter estimation . . . 44
- 3.6 Experiment results . . . 46
  - 3.6.1 Experiment setup . . . 46
  - 3.6.2 Results . . . 48
    - 3.6.2.1 Tradeoff between effectiveness and efficiency . . . 48
    - 3.6.2.2 Analysis of query latency distribution . . . 52
    - 3.6.2.3 Relationship to other retrieval models . . . 54

4 Ranking under Temporal Constraints 57
4.1 Ranking with time constraints . . . 59
4.1.1 Linear ranking functions . . . 59
4.2 Constrained linear ranking functions . . . 61
4.2.1 Prediction models . . . 62
4.2.1.1 Independent Prediction Model . . . 63
4.2.1.2 Joint Prediction Model . . . 66
4.2.2 Temporal constraint enforcement . . . 70
4.2.3 Parameter estimation . . . 71
4.3 Experiments . . . 73
4.3.1 Effectiveness vs time constraints . . . 74
4.3.2 Satisfying time constraints . . . 77
4.3.3 Expected effectiveness across constraints . . . 80
4.3.4 Comparison with SD model . . . 83

5 A Cascade Ranking Model for Efficient Ranked Retrieval 85
5.1 Cascade model . . . 91
5.1.1 Pruning functions . . . 94
5.2 Learning the cascade . . . 97
5.2.1 Cost estimation . . . 98
5.2.2 Tradeoff metric . . . 99
5.2.3 Learning . . . 100
5.2.4 Cascade Stage Construction . . . 102
5.2.5 Analysis . . . 104
5.3 Experiments . . . 107
5.3.1 Experiment Setup . . . 108
5.3.2 Implementation Details . . . 109
5.3.3 Effectiveness vs. Efficiency . . . 110
5.3.4 Cascade Analysis . . . 113
5.3.5 Parameter Variations . . . 115

6 Constrained Ranking via Cascades 119
6.1 A Constrained Cascade Model . . . 120
6.1.1 Constrained cascade . . . 121
6.1.2 Learning constrained cascade models . . . 125
6.2 Experiments . . . 130
6.2.1 Experimental setup . . . 130
6.2.2 Results . . . 131
6.2.2.1 Ranked effectiveness vs time constraints . . . 131
6.2.2.2 Ensuring time requirements . . . 134
6.2.2.3 Document pruning and pruning loss . . . 136
6.2.2.4 Average effectiveness across time requirements . . . 137

7 Conclusion 140
7.1 Limitations 140
7.2 Future work 141

Bibliography 143

List of Figures

1.1 Tradeoff between ranking model effectiveness and efficiency 2
1.2 Search quality vs Efficiency and Scalability. 3
1.3 Basic setup of learning to efficiently rank. 4

3.1 Ranking model effectiveness/efficiency tradeoff. 26
3.2 Efficiency function definitions. 34
3.3 Distribution of query execution time. 53

4.1 MAP versus time for Indep, Joint and QL models on title and description queries in the test sets of Wt10g, Gov2 and Clue. 75
4.2 The fraction of query evaluation times that satisfy the imposed time constraint. 78

5.1 An example cascade. 90
5.2 NDCG20 as a function of time. 116

6.1 NDCG20 and P20 versus time requirements. 132
6.2 Bar chart showing the fraction of query evaluation times that satisfy the imposed time constraint for Wt10g, Gov2, and Clue. 135
6.3 % documents pruned as a function of time constraints, and the corresponding pruning loss (insert). 136

Chapter 1

Introduction

1.1 Motivation

Information retrieval (IR) is founded on constructing ranking (i.e., retrieval) models for finding and ranking relevant information to user queries. IR models have been the focus of much research over the past several decades, and advances in this area have led to many practical systems that enable us to better analyze and search for information. For example, Web search, one of the most important applications of IR, is a daily activity for many people. Search personalization and information filtering have become increasingly popular, particularly on eCommerce web sites.

Typically, from the end user's point of view, two factors are important in large-scale IR. First, the retrieved information should match the user's information needs to a high degree (i.e., relevance). Second, results should be returned to the user in a timely manner (i.e., efficiency). However, effectiveness and efficiency are inherently inter-connected and competing forces that often counteract each other. As illustrated in Figure 1.1, the quest for effectiveness, which has been the central focus in learning ranking models, can lead to very complex and slow ranking models, completely impractical for large-scale datasets; while the converse is true for simple ranking models. Despite the fundamental tradeoff between these two competing measures, for much of the history of academic research on learning ranking models,

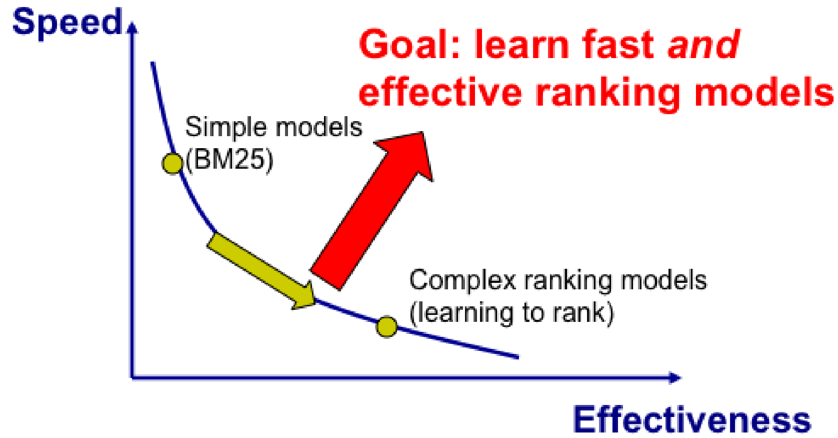


Figure 1.1: Tradeoff between ranking model effectiveness and efficiency.

explorations in effectiveness and efficiency have largely been disjoint. As shown in Figure 1.2, on one side there are researchers who develop highly effective, yet practically infeasible models en masse, by engineering sophisticated ranking features and ranking models [1, 2, 3, 4, 5]. On the other side of the dichotomy are the researchers who make existing models faster and more scalable, either by designing fast and approximate query evaluation strategies [6, 7, 8] (which typically comes at a cost of degraded effectiveness), or spending more system resources to counter-balancing relevance gains (e.g., buying more hardware, implementing caching [9], or spending additional effort in low-level optimizations).

Our work synthesizes these two equally important threads of research and introduces a unified learning-based framework that accounts for tradeoffs between effectiveness and efficiency, in the context of learning highly effective *and* highly efficient ranking models for large-scale datasets. At a basic level, our framework builds ranking models whose speed and accuracy can be explicitly controlled. In

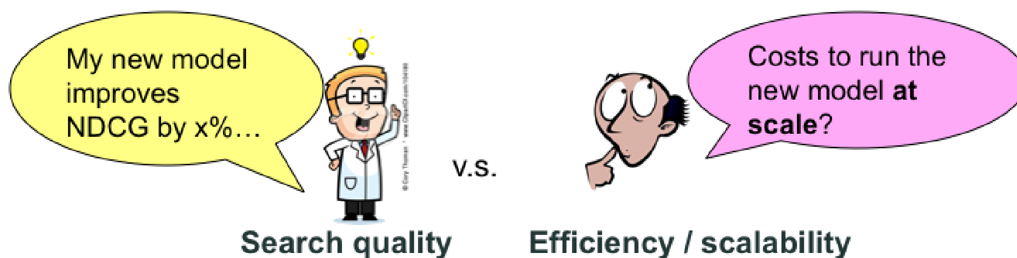


Figure 1.2: Search quality vs Efficiency and Scalability.

contrast to previous approaches for creating efficient large-scale systems, which typically involve spending extra system resources for effectiveness gains, or creating approximated answers which may have degraded effectiveness, we introduce new classes of models and learning algorithms that can *improve* ranking effectiveness *without* incurring extra costs. We achieve this by deriving *both* efficiency and effectiveness from the ranking model itself. As a result, the new models are capable of overcoming the tradeoff curse and reside in a more optimal region in the tradeoff space with high effectiveness and high efficiency (Figure 1.1).

1.2 Overview of learning to efficiently rank

In this section, we discuss our problem setup and outline the proposed techniques for constructing fast and effective ranking models.

1.2.1 Problem setup

The basic setup we consider is similar to the standard supervised learning to rank. As shown in Figure 1.3, given a set of training data, which consists of training

queries, documents, and relevance labels, we would like to learn a ranking model to optimize a given IR metric. A key difference with respect to effectiveness-based learning is that our framework takes model run-time costs as an input (estimated by a cost model), and use that information to help derive a cost-effective ranking model during learning. Therefore, our setting is orthogonal to system-engineering aspects of efficiency (caching, document partitioning, etc.) – their effects on query response time are captured by the cost model, simply served as input to model learning.

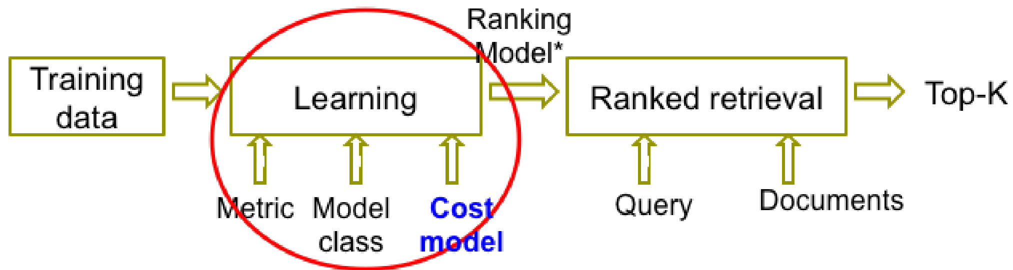


Figure 1.3: Basic setup of learning to efficiently rank.

Our proposed framework can be more easily understood in the context of multi-objective learning, where the goal is to devise machine-learned models that optimize multiple objective metrics. For example, in the context of learning to rank, previous studies have considered optimizing models for freshness and effectiveness [10], as well as optimizing for a primary evaluation metric (e.g., NDCG) and a secondary evaluation metric (e.g., user-click profiles) [11]. While many problems belong to the general family of multi-objective learning, we note that each problem instance requires their own solutions since their optimization objectives are different.

There are three broad challenges in learning to efficiently rank. First, previous models are either effectiveness-centric or efficiency-centric. We need to design new

models that have the ability to be simultaneously fast and effective. Second, what is the optimization metric for model learning? Effectiveness metrics have been well studied. However, how to define the efficiency and tradeoff metrics is still an open question. Finally, given the new class of models and optimization metrics, model learning is an interesting problem – it requires two competing metrics to be jointly optimized, unlike the previous effectiveness-based learning. We now give a brief overview on how we tackle these issues in the context of linear ranking models.

1.2.2 Constructing efficient and effective linear models

A great number of retrieval models have been proposed over the years, from the unigram language models [12, 13, 14] to the more complex models that use a variety of ranking features [2, 15, 16, 17, 18]. In this work, we consider the class of linear-feature based models, where the relevance score assigned to each document can be expressed in terms of a linear combination of feature values computed from each query-document pair. We associate an efficiency and an effectiveness dimension with each model and query pair. The efficiency of the model can be measured in terms of average query execution time, or the number of total documents over which the retrieval model computes scores in order to answer a query, and effectiveness of the model can be calculated in terms of standard retrieval metrics such as mean average precision [19] or early-precision [19], which measure the model's ability to return relevant results early in the ranked list.

The main difference between various linear-feature based models is the type

of features they employ. Features used in simple models (such as unigram language models [12, 13]) are cheap to compute at query-time, although the oversimplified feature space may lead to sub-optimal effectiveness. Features used in complex models (such as term dependence models [15, 20]) are more effective in general, albeit may be expensive to compute at query-time. As a result, simple models are better in terms of efficiency, but (in general) worse in terms of effectiveness as compared to complex models, and the opposite is true for complex models. In between these two extremes lie other ranking functions that encode a variety of efficiency/effectiveness tradeoffs. By learning an efficient retrieval model, we mean selecting an optimal ranking model from the large array of ranking models to satisfy a given requirement on relevance and speed from the user.

In this work, we use both term-based [12, 13] and term-proximity features [21, 15]. We assume the term-based features are freely available and term-proximity features are unindexed (which requires query-time computations). This feature pool is used to model differential query-time feature costs and allows us to explore the different efficient ranking models that encode a spectrum of speed/effectiveness tradeoffs.

Broadly, we are interested in the following problems for building ranking models for Web-scale document collections:

1. Improve the *average* query execution time of user submitted queries while not degrading their average effectiveness too much.

2. Produce high quality results within an efficiency (time) requirement specified for *each* query. Since each query may have a different time requirement, the

ranking function for each query can be very different.

3. Return the best top-K documents for a query such that top-K ranked effectiveness and retrieval efficiency are jointly optimized.

4. Return the best results under time constraints such that effectiveness and retrieval efficiency are jointly optimized.

In the following chapters, we formally state these problems and describe our approaches for solving them. For now, we note that the retrieval time is mainly governed by the complexity of the ranking model, as well as the size of the document collection. Our proposed solutions for the first two problems rely on reducing the complexity of the ranking models in order for them to work efficiently (and effectively) on the entire document collection. We propose to improve efficiency by selectively pruning features in a query-dependent manner (i.e., removing query-dependent features that do not contribute much to effectiveness but incur high complexity). For the first problem, we propose a family of tradeoff metrics between efficiency and effectiveness, and we introduce the notion of *efficient linear models*, which augment the conventional linear feature-based models with a pruning threshold defined over query-dependent features. Both the feature weights and pruning threshold in the efficient linear model are learned to optimally balance retrieval effectiveness and efficiency according to a given tradeoff metric. We then empirically evaluate our proposed models under various tradeoff scenarios, to demonstrate their ability to adapt to these different tradeoffs. We also compare the proposed models against several baseline models, including a state-of-the-art linear model, in terms

of retrieval effectiveness, efficiency, and their tradeoff.

For the second problem, we aim to construct the most effective ranking model for each query such that the retrieval efficiency of the model meets the query's efficiency (time) requirement with high likelihood. This requires us to build a ranking model at query-time for each query. For this problem, we first introduce the notion of *temporally constrained linear model*, which is a linear model that is restricted to returning results within a pre-specified time constraint. We then propose two methods for constructing effective temporally constrained linear models for given user queries and their associated time requirements. Both methods incur minimum overhead for the online construction – $O(|q|log|q|)$, where $|q|$ denotes the query length. This is trivial as compared to query execution time. We then empirically evaluate the two temporally constrained ranking algorithms across a wide range of efficiency requirements to demonstrate the resulting temporally constrained models can achieve high effectiveness across time requirements and can return more effective results than several commonly studied ranking functions under the same time constraint.

For the third problem, we consider top-K retrieval over web-scale document collections, and are interested in returning the best top-K documents such that the ranked effectiveness and retrieval efficiency are jointly optimized. We observe that in Web search, users will only browse through a very small number of returned documents, e.g., K=20. Furthermore, the number of relevant documents for each query is usually very small as compared to the Web collection size. In this case, applying a complex yet effective ranking model on the entire collection may not be desirable

for efficiency reasons, since most documents evaluated are likely to be non-relevant and/or outside of the top-K. This motivates a cascade model for top-K retrieval, which uses a sequence of ranking models, ranging from low to high complexity, to progressively filter and re-rank documents, with the goal of obtaining high top-K ranked effectiveness at a much lower cost as compared to effectiveness-centric models. A large number of unlikely documents can be pruned early with simple/cheap models, so that more complex ranking models can be used on a small number of documents for improving top-K ranked effectiveness. We present the cascade model structure and parameter space, then discuss the cascade model learning problem.

For the fourth problem, we synthesize ranking under temporal constraints and cascade-based ranking functions. Given a time budget, a constrained version of the cascade model is automatically constructed to return the best possible ranked list within the specified time limit. The constrained cascade aims to simultaneously optimize both the features used within the model and the document refinement strategy. Thus, the constrained cascade can better utilize the available budget, via the stage-wise document pruning mechanism of the cascade model, yielding a more robust, more efficient, and more effective retrieval model.

1.3 Contributions

The following is a summary of our primary contributions:

1. **Learning to efficiently rank.** We propose the *learning to efficiently rank* framework, which takes an efficiency-minded look at building effective rank-

ing models given a pre-specified requirement on efficiency/effectiveness. This proposed framework represents a significant departure from the traditional ranking models and "learning to rank", since it *jointly* optimizes the effectiveness and efficiency of a ranking model. The framework could be used in real-world applications which require the ranking model to adapt to different user efficiency/effectiveness requirements. It can also be used for query load balancing due to its ability to control query execution times.

2. **Efficient linear model.** As an instance of "learning to efficiently rank", we propose a new class of ranking models called *efficient linear models* and a class of tunable tradeoff metrics between retrieval efficiency/effectiveness. The new class of models can be learned to optimize the desired tradeoff to achieve an optimal balance between retrieval efficiency and effectiveness.

3. **Temporally constrained ranking functions.** As another instance of "learning to efficiently rank", we propose novel methods based on probabilistic graphical models to automatically infer optimal ranking functions that are both highly effective and capable of satisfying efficiency requirements on a *per-query* basis with high likelihood.

4. **Cascade ranking functions.** As another instance of "learning to efficiently rank", we discuss a novel cascade ranking model for top-K retrieval over web-scale collections. In contrast to the previous two cases, the ranking model progressively reduces the candidate document set size to a query with the goal of obtaining high top-K ranked effectiveness at a much lower cost.

5. **Constrained cascade ranking functions.** As another instance, we propose a constrained version of the cascade that returns results within time budgets by simultaneously reducing document set size and unnecessary features.

1.4 Outline of chapters

The rest of the dissertation is organized as follows:

- Chapter 2 reviews related work and background.
- Chapter 3 presents our solution for learning an efficient model that optimizes a given effectiveness/efficiency tradeoff. We propose optimization metrics as well as a framework for constructing such functions.
- Chapter 4 describes our solution for temporally constrained ranking, which return the most effective results under a time budget for *each* query and satisfies the actual time requirements (at query-time) with high likelihood.
- Chapter 5 presents a cascade ranking model for top-K retrieval. We first discuss related work and put our work in context, then discuss the cascade model, and state the cascade learning problem and solutions.
- Chapter 6 presents a constrained version of the cascade model for returning results within time constraints.
- Chapter 7 concludes and points out future work directions.

Chapter 2

Related Work and Background

2.1 Ranking models

Ranking is a central problem in information retrieval, and it is widely-studied in many different contexts such as in image retrieval [22], document retrieval [19], and speech retrieval [23]. In this work, we focus on document retrieval. Given an indexed document repository, when a user issues a free-text search query, the role of a ranking model is to retrieve relevant documents for the given query and present them in a ranked list format to the user, where the documents in the ranked list are ordered by their estimated degrees of relevance to the query (most relevant at the top). While there exist many classical models for performing this task, such as the vector space model [24], Boolean model [25], and the unigram language model [12, 13], there are several limitations that they face. Many of these models depend on manual parameter tuning, which may not result in optimal parameter settings, especially when there are many parameters. Additionally, most of these models only use simple term-occurrence features, which count the occurrences of individual query terms in each document. Although a lot of other useful ranking features exist, such as term proximity features and page importance features, manually incorporating them into the basic framework of these classical models is cumbersome, if not impossible.

Machine learning is an effective tool to automatically tune parameters and

combine many useful features in a principled way into the ranking model. "Learning to Rank" [2] is a recently emerged research area in information retrieval, which uses machine learning methods to train ranking models for search effectiveness. Broadly, three categories of learning to rank algorithms have been proposed: the pointwise approach [26, 27], the pairwise approach [28, 29, 30], and the listwise approach [31, 32, 33]. The pointwise approach assumes that each query-document training pair has a numerical score, and the learning to rank problem is formulated as a regression problem – predict the score of the document given the query-document pair. Many existing ordinal regression and classification algorithms can be readily applied to this pointwise approach. However, ranking is an ordering problem rather than a classification/regression problem – we are more concerned with the relative ordering of the documents rather than their absolute document scores. To handle ground truths of pairwise preferences, the second category of algorithms (pairwise approach) formulates ranking as a binary preference classification problem – a binary classifier is learned to tell which document is better for any given pair of documents. The loss function used by this approach is usually defined in terms of the average number of preference inversions. To further handle the ground truth in terms of partial/total orderings, the listwise approach aims to directly optimize the ranking measures, averaged over training queries. However, it is well-known that commonly-used ranking measures (mean average precision, precision-at-10, etc.) are not continuous functions with respect to a ranking model's parameters, which makes them difficult to work with from an optimization point of view. Thus, approximations or bounds on evaluation measures are typically used for optimization purposes.

Another thread of learning to rank work has focused on linear feature-based models [15, 20, 34, 35, 36, 37]. Linear feature-based models are a class of simple, yet very effective ranking functions. Many widely used ranking models belong to this family of ranking functions and they have demonstrated effectiveness over publicly benchmarked retrieval tasks [38, 39]. A linear model scores each document in response to a query via a weighted linear combination of query-document features, and the document scores induce a total ordering of the documents. The main advantage of linear feature-based models is their ability to combine various kinds of ranking features, such as unigram features [12], term proximity features [15], and linguistic features [35], in a straightforward manner. The model parameters (i.e., feature weights) can be estimated through learning to rank methods [40, 33, 41].

An underlying assumption of all aforementioned methods is that the ranking function for retrieval has been specified, and efficiency is improved by speeding up query evaluation for the ranking function. However, an important fact about search queries is that they are inherently ambiguous, and are quite different from other query languages which are governed by strict syntax or semantics rules (e.g., SQL queries). Due to this ambiguity, *different* ranking functions can be used for the *same* user query, where the ranking functions can vary greatly both in terms of how they interpret the user query (e.g., whether or not query terms are independent or follow certain dependency structure), and in terms of their efficiency characteristics (e.g., ranking functions that employ complex features are generally slower, although can be more effective, than simple ranking functions).

As a result, we observe that selecting an optimal ranking function for a user

query, from the space of all possible ranking functions, should depend on how well the ranking function's effectiveness and efficiency characteristics match with the requirements of the user. For example, while most users may want their search results immediately, others may not mind waiting a little extra time if it means their results would be better. This same idea can be applied to information needs. Certain classes of simple queries are expected to be answered immediately. However, for complex information needs, users may be willing to have 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, rather, the ranking function should be customized to meet user expectations in retrieval relevance and speed. In other words, while learning to rank gives rise to highly effective ranking models, the models' high complexity limit their practical applications to web-scale data. As a result, these models are usually evaluated over a small set of "toy" data for effectiveness alone [2]. In our work, the goal is to design practical and efficient ranking models that work well on web-scale data collections. For this purpose, we look at the empirical performance over large datasets (speed and ranked effectiveness) of the proposed ranking models to ascertain their practical value. "Learning to rank" (i.e., only optimizing effectiveness) is actually a special case of our proposed "learning to efficiently rank" framework when the efficiency requirement is relaxed.

2.2 Query evaluation and caching strategies

The previous section discusses related work on how to learn a ranking model; in this section, we discuss given a ranking model and a user query, how to evaluate the query given the ranking model, i.e., query evaluation. Many optimization techniques have been proposed for efficiently evaluating user queries. In this section, we touch upon several of the most commonly-used techniques for query processing – early exit strategies [6, 7, 8], caching [9], index pruning [42, 43, 44, 45, 46] and query segmentation [47].

While the starting point of the work to be reviewed in this section (query evaluation, caching, and index pruning/query segmentation) and learning to efficiently rank is the same – obtaining faster query execution time, we want to point out these techniques and our work are two *complementary* approaches for improving efficiency in search. Query evaluation and caching are concerned with how to evaluate a user query *given* a ranking model, e.g., through query-processing techniques such as early termination [6, 7, 8]. However, we learn ranking functions, under a given query evaluation strategy and caching strategy. More specifically, the retrieval efficiency of a candidate ranking function is estimated through a *cost model*. The caching and query evaluation strategy will impact the estimated retrieval efficiency of a given ranking function. In this work, we assume that the candidate ranking functions under consideration will be run on a *common* retrieval system, i.e., the same query evaluation/caching strategy will be used for the candidate ranking functions, therefore, their retrieval efficiencies can be directly obtained from the cost

model specifically built for the given retrieval system. There are various ways for a cost model to characterize the efficiency of a ranking function (used within a retrieval system), such as directly measuring it in terms of query execution time, or using analytical estimators [48]. In our work, we use both techniques for measuring efficiency (for both training and test queries). Finally, the estimated retrieval efficiencies of ranking functions will be utilized by our learning to efficiently rank framework for selecting a ranking function that jointly optimizes efficiency and effectiveness according to a given metric.

2.2.1 Early exit strategies

There are two flavors for query evaluation in IR systems: exact [49] and inexact top-K retrievals [6, 7, 8]. Since free-text search queries are inherently ambiguous, retrieving the exact top-K documents according to a ranking function may not necessarily be the K best for the query, because the ranking function is only a proxy for the user's perceived relevance [19]. Thus, most practical query evaluation algorithms belong to the category of inexact top-K retrievals, which is the focus of our discussion here. Much research has been devoted to early exit strategies for conventional IR ranking models (those that are not machine learned) [7, 45, 50]. The main idea behind this thread of work is to only traverse a subset of the documents in the postings lists associated with query terms to speed up query evaluation. The classical approach is to only traverse through the postings lists associated with the important query terms (as determined by their inverse document frequency values,

etc.), and to ignore postings lists for the remaining query terms [50]. Alternatively, for the remaining query terms, their postings lists can be traversed except that the documents not encountered earlier in the postings lists of high-impact query terms are ignored [50].

Recent work on early exit strategies for machine learned ranking functions [6] considers the problem of accelerating query evaluation for a fixed ranking function through short-circuiting score computations in additive learning systems. More specifically, a complex ranking function in the form of additive ensembles (e.g., boosted decision trees [51]) is learned during the training phase. At query time, each document goes through the entire chain of scorers to retrieve a partial contribution to its final score from each of the individual scores. Efficiency is important if the number of scorers is high. Thus, short-circuiting techniques in the form of terminating score computations for documents that are unlikely in the top-K ranked list are proposed to speed up the scoring process [6]. In addition, Cambazoglu et al. [6] focus on optimizing the query evaluation process given a particular additive ensemble–and not about learning the ensemble. In contrast, we focus on *learning* an end-to-end ranking system to optimize a desired efficiency/effectiveness tradeoff metric. We formally introduce a novel boosting algorithm for learning a cascaded system in Chapter 5.

It is interesting to note that although the main idea behind query evaluations for both machine learned ranking functions and conventional ranking functions is the same – compute top-K documents as fast as possible without sacrificing the results quality, a noticeable difference is that for machine learned ranking functions,

score contributions generated by the complex scorers are not known before query-time [6], whereas for conventional IR ranking models, such information is already stored in the document index.

2.2.2 Caching

Another way to speed up query evaluation is through caching [9]. There are two possible ways to implement caching – caching query results and caching query term postings lists. For caching query results, as the search engine computes results for a given query, it may decide to store the answers in memory for future queries [9]. For caching query term postings lists, the postings lists associated with a query may be stored in memory for future use. Although returning an answer to a query that already exists in cache is much more faster than computing results using the cached postings lists, the previously unseen queries occur much more frequently than previously unseen terms, which means that a higher miss rate for cached answers [9]. As a result, most work has focused on caching postings lists. The main challenge is to determine what postings lists to cache for a certain amount of available memory. A natural way to do this is to store the postings lists of frequently occurring query terms, while trying to avoid query terms with long postings lists (because they will take up more memory).

As mentioned earlier, the caching or query evaluation strategy will only impact our cost model for estimating the retrieval efficiency of a given ranking function, and we assume the candidate ranking functions will be run on a common retrieval system,

so the same query evaluation/caching strategy will be used for them.

2.2.3 Index pruning/query segmentation

Our problem is quite different from previous work in index pruning [42, 43, 44, 45, 46] and query segmentation [47]. The primary goal of index pruning is to create a compact index offline and search over this index to gain better efficiency. In query segmentation, a syntactic parser is used to identify noun phrases in a query, and only the features defined over the noun phrases, rather than all features, are used to retrieve documents. This technique can be especially beneficial for verbose queries since there may be more imprecise terms or terms that do not contribute to the user's actual intent in such queries; using these terms will incur high computational cost and they may not help retrieval effectiveness too much. While both index pruning and query segmentation 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.

2.3 Coarse-to-fine models

The cascade model described in Chapter 5 is related to the general class of coarse-to-fine models that have been used in real-time object detection and classification problems in computer vision and machine learning. For instance, in the problem of real-time face detection in images [52], a sequence of binary classifiers

of increasing complexity are applied to progressively filters the object images, such that the mostly unlikely images are rejected early by simple classifiers, and the more promising object-like regions are given to the more complex classifiers for further consideration. A key difference between their work and ours is that our problem is a ranking problem, rather than a classification/detection problem. Thus, we are not interested in getting high recall or precision as [52]. We are interested in the *top-K ranked effectiveness*, which only depends on how the top-K documents are *ranked*. This is in line with "learning to rank" for information retrieval [2, 53, 28, 54], where the goal is to construct the best ordering of documents, rather than predicting correct document scores or labels. Several challenges will arise from optimizing the cascade system with respect to the top-K ranked effectiveness, which we discuss Chapter 5.

Another example of coarse-to-fine models is the structured prediction cascade [55]. Structured prediction poses a tradeoff between the need for highly complex models for better predictive power and the limited computational resources for inference [55]. A structured prediction cascade was formulated to progressively filter the space of possible outputs, and due to the filtering on the output space, inference complexity can be significantly reduced. Given this, a key difference between structured prediction cascade and our work is that we *learn* the structure of the cascade, i.e., what ranking models are used to construct the cascade, as well as the parameters of these ranking models; whereas in [55], the structure is assumed to be given, and the goal is to learn the model parameters only. Another difference is ranking effectiveness is measured by top-K ranked effectiveness (e.g., normalized

discounted cumulative gain, etc.), which is a common measure used by web-search applications.

2.4 Other work that goes beyond effectiveness

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 [56]. 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 [57] 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 imposes a penalty for including common terms in the expanded query, since such

terms are likely to increase query execution time. This was the first effort that we are aware of that modeled efficiency in a search engine-specific manner. However, our work focuses on ad hoc retrieval, as opposed to query expansion. Furthermore, the convex program is solved at query-time in [57]. The issue of the how such a program can be solved efficiently at query-time was not discussed. In our work, we also need to deal with the issue of constructing a ranking model at query-time – for instance, for the problem of *ranking under temporal constraints*, we build a query-specific ranking function to satisfy the given query's time requirement, and as we show (Chapter 4), our technique for model construction at query time incurs minimal overhead – $O(|Q|log|Q|)$, where $|Q|$ denotes the query length.

Our work is also related to optimizing multiple *effectiveness* metrics [11, 10, 58]. Svore *et al.* [11] use a standard web relevance measure, NDCG, as the primary optimization metric and a relevance derived from click-data is used as the secondary metric; the performance of the primary measure is maintained constant while the algorithm tries to improve the secondary measure. Dai *et al.* [10] develop a multi-objective learning to rank algorithm for freshness and relevance by extending a state-of-the-art divide and conquer ranking approach [59]. A difference between these and our work is that we treat multiple measures, which are potentially competing, as first-class metrics during optimization, unlike the "tiered" approach [11]. In addition, instead of capturing just another facet of web search [11, 10], our efficiency metrics evaluate the speed of the ranking model. Model robustness, as another optimization objective, has recently been studied by [58]. However, our focus here is on optimizing for model speed and effectiveness, rather than robustness.

Another line of related work is query-dependent models. This line of work resulted in recent papers that consider query-specific loss functions [60], where queries of different types (e.g., navigational, information) are optimized with different loss functions. Geng *et al.* [61] proposed a k-Nearest Neighbor based method which trains a query-dependent ranking function for each query based on its nearest neighbors in the training set. Bian *et al.* [59] proposed a clustering-based divide-and-conquer approach for building query-dependent ranking models. However, none of these methods considers model efficiency nor robustness issues. Our proposed framework generalizes and complements this thread of work by learning cost-sensitive (and robust) query-dependent models.

Chapter 3

Efficient Linear Models

A lot of ranking models have been proposed over the years for retrieval effectiveness. They range from the very simple term occurrence-based vector space model and language model to the more complex linear feature-based model that can use a large number of ranking features. Many of the more recently proposed models are situated in the context of "learning to rank" – given a collection and a set of training queries and relevance judgement, machine learning techniques are used to devise ranking models to optimize retrieval effectiveness over the training data. This thread of work has focused entirely on effectiveness. As a result, there is no well-established a way to balance effectiveness and efficiency in these models. As shown in Figure 3.1, a tradeoff exists between effectiveness and efficiency amongst these models – the most effective models are generally the most expensive ones (e.g., long query execution time), and it would be entirely impractical to directly apply them to web-scale document collections, even if that means high quality results. Furthermore, in reality, both user and query efficiency requirements are diverse, optimizing models for effectiveness alone ignores such diverse efficiency requirements altogether.

In contrast to effectiveness-centric frameworks, we introduce "learning to efficiently rank" in this and subsequent chapters, which considers the tradeoff between

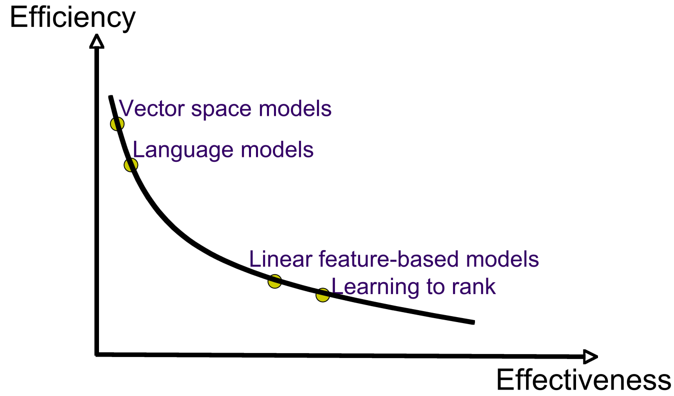


Figure 3.1: Ranking model effectiveness/efficiency tradeoff.

retrieval effectiveness and efficiency in the context of developing highly effective and efficient ranking models. Since linear feature-based models form the basis of our proposed efficient models, what features used in these models will play a large part in determining retrieval efficiency, because the query-document features may need to be evaluated at *query-time*. Broadly, we propose three novel approaches for improving efficiency. The starting point of the first two approaches (this and the next chapters) rely on reducing the complexity of a linear feature-based ranking function through query-dependent feature elimination/selection, so that when applied to ranking large-scale document collections, the query execution time can be significantly reduced (i.e., by reducing the number of query-document features to be evaluated at query-time). The third and fourth approaches (Chapters 5 and 6) rely on using a novel cascade ranking model to identify the ranked results for each query in a fast and effective manner, by progressively reducing and re-ranking a number of candidate documents. In comparison to the first two approaches, they potentially lead to both high efficiency (by reducing document set size) and high top-K ranked effectiveness (by using highly complex and effective models towards the end of the

cascade). In this and the next chapter, we focus on the first two problems:

1. Devise a ranking model which optimally balances retrieval efficiency and effectiveness according to an optimization tradeoff metric (this chapter);

2. Devise the most effective ranking model for a given temporal constraint imposed for *each* query (Chapter 4).

The main difference between these scenarios is that in the first case we consider selecting an optimal operating point in the space of effectiveness/efficiency tradeoffs (Figure 3.1) according to an optimization metric that characterizes the desired tradeoff, but there is no need to provide control on an individual query basis. Whereas in the second case, we devise temporally constrained ranking models for each query, where the models should satisfy query-specific time requirements.

An important observation is that efficiency is a query-time measure (since it depends on the actual query), thus, our efficient ranking models need to be constructed at query-time (rather than offline) to account for various tradeoff scenarios between efficiency/effectiveness. Therefore, the online construction of such ranking models need to be sufficiently fast. As we will show, our proposed solutions incur negligible overhead for constructing these ranking models.

3.1 Background on Linear Feature-based Models

In this section, we give an overview of linear feature-based models, since they form the basis of our proposed efficient ranking models. The main difference between

different linear models is the types of features they employ and how the feature weights are computed. We first describe the general form of linear feature-based models, and then focus on two particular linear models — the sequential dependence (SD) model [15] and the weighted sequential dependence model (WSD) [34].

3.1.1 Linear feature-based models

Under linear feature-based ranking models, the relevance score assigned to each document is computed by a weighted linear combination of feature values computed from the query-document pair. The features can be divided into two broad classes: 1) query-dependent features (a.k.a. query-document features), such as the number of occurrences of each query term in the document. These features may need to be evaluated at *query-time*, which will contribute to query latency; and 2) document features, such as PageRank and spam score, which are only specific to the document. Many widely used ranking models belong to the family of linear feature-based ranking functions [15, 20, 34, 35, 36, 37], and they have demonstrated effectiveness over several publicly benchmarked retrieval tasks [38, 39].

More specifically, a linear ranking function is characterized by a set of features $F = f_1, \ldots, f_N$ and the corresponding model parameters $\Lambda = \lambda_1, \ldots, \lambda_N$. Each feature f_i is a function that maps a query-document pair (Q, D) to a real value[1]. The relevance of document D with respect to query Q is computed as:

$$Score(Q, D) = \sum_i \lambda_i f_i(Q, D) \tag{3.1}$$

[1]Note if the feature is document feature, its value is independent of the query.

Hence, as mentioned earlier, each document is scored by a weighted linear combination of the feature values computed over the document and query. The scoring function induces a total ordering on the document set for each query. One main advantage of linear feature-based models is that they provide a way to utilize many different kinds of features in a straightforward manner. Examples include term-occurrence features [12, 13], term proximity features [15], and linguistic features [35], amongst others. Many learning to rank approaches can be used for learning a linear ranking function (i.e., estimating w) such as SVM [40], SVM^{MAP} [33], and coordinate level ascent algorithms [41].

3.1.2 SD and WSD models

From the general formula (Eqn 3.1), different linear models can be instantiated by specifying their corresponding feature sets and the associated weights. Among the possible linear model instantiations, we restrict our attention to the *sequential dependence model* (SD) [15] and the *weighted sequential dependence model* (WSD) [34], since they are related to our efficient sequential dependence models (ESD) to be introduced later in this thesis. Both of SD and WSD models employ a combination of term-based and term proximity-based ranking features to score documents. The features used by SD and WSD models is listed in table 3.1. Specifically, for features they use the number of occurrences of each query unigram in a document, the number of occurrences of exact phrase "q_j q_{j+1}", where q_j q_{j+1} denote a query term bigram, and the number of occurrences of unordered window q_j q_{j+1}

$$f_{T,Dir}(q,D) = \log\left[\frac{\mathrm{tf}(q,D)+\frac{\mu}{|C|}\mathrm{cf}(q)}{|D|+\mu}\right]$$

$$f_{T,BM25}(q,D) = \frac{(k_1+1)\cdot\mathrm{tf}(q,D)}{K+\mathrm{tf}(q,D)}\log\left[\frac{N-\mathrm{df}(q)+0.5}{\mathrm{df}(q)+0.5}\right]$$

$$f_{O,Dir,S}(q_j,q_{j+1},D) = \log\left[\frac{\mathrm{tf}(\mathrm{OD}(S,q_j,q_{j+1}),D)+\frac{\mu}{|C|}\mathrm{cf}(\mathrm{OD}(S,q_j,q_{j+1}))}{|D|+\mu}\right]$$

$$f_{O,BM25,S}(q_j,q_{j+1},D) = \frac{(k_1+1)\cdot\mathrm{tf}(\mathrm{OD}(S,q_j,q_{j+1}),D)}{K+\mathrm{tf}(\mathrm{OD}(S,q_j,q_{j+1}),D)}\log\left[\frac{N-\mathrm{df}(\mathrm{OD}(S,q_j,q_{j+1}))+0.5}{\mathrm{df}(\mathrm{OD}(S,q_j,q_{j+1}))+0.5}\right]$$

$$f_{U,Dir,S'}(q_j,q_{j+1},D) = \log\left[\frac{\mathrm{tf}(\mathrm{UW}(S',q_j,q_{j+1}),D)+\frac{\mu}{|C|}\mathrm{cf}(\mathrm{UW}(S',q_j,q_{j+1}))}{|D|+\mu}\right]$$

$$f_{U,BM25,S'}(q_j,q_{j+1},D) = \frac{(k_1+1)\cdot\mathrm{tf}(\mathrm{UW}(S',q_j,q_{j+1}),D)}{K+\mathrm{tf}(\mathrm{UW}(S',q_j,q_{j+1}),D)}\log\left[\frac{N-\mathrm{df}(\mathrm{UW}(S',q_j,q_{j+1}))+0.5}{\mathrm{df}(\mathrm{UW}(S',q_j,q_{j+1}))+0.5}\right]$$

Table 3.1: Features used in our linear ranking functions. 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, $|D|'$ is the average document length in the collection, and $|C|$ is the total length of the collection. N and N' are the window sizes for ordered and unordered phrases, respectively, where $N \in \{1, 2, 4\}$ and $N' \in \{2, 4, 8\}$. Finally, μ is a Dirichlet feature scoring function hyperparameter, k_1 and b are BM25 feature scoring function hyperparameters.

(window span = 8) in a document.

In the case of SD model, the weight on a particular feature f_i depends on the *type* of feature f_i. For instance, most commonly, all term occurrence features used by SD receive a weight of 0.82, and the term proximity features (e.g., exact phrase, unordered window features) each receives a weight of 0.09; these weights reflect best-practice settings and have been used in several previous work [15, 20, 34].

An advantage of the parameter tying between features of the same type is simplicity in ranking model. A drawback is that it cannot differentiate between features defined over "important" query concepts[2] from features defined over less important

[2]In this work, a query concept simply refers to a unigram q_j or a bigram $q_j\ q_{j+1}$ in the query.

Concept-importance features	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 3.2: Concept-importance features (a.k.a. meta-features) used in ranking.

query concepts, since they will be assigned the same weight in the ranking model if they are in the same feature class. For instance, for the query "Shenandoah Valley Tourist Attractions", the query concepts "Shenandoah Valley" and "Valley Tourist" clearly have different importance in the query, such that "Shenandoah Valley" may have higher importance than "Valley Tourist". However, the SD model treats them as equally important (by assigning their term proximity features the same weight in the ranking model). The *weighted sequential dependence model* (WSD) [34] was proposed to improve the SD model by letting the feature weights (for query-dependent features) depend on the query concepts, such that features are assigned weights in accordance to the importance of the concepts that they are defined over. Formally, let e_i denote the query concept that feature f_i is defined over, the weight λ_i of feature f_i takes a parametric form:

$$\lambda_i(e_i) = \sum_j w_j \; g_j(e_i)$$

where g_j's are the concept-importance features (a.k.a. meta-features) defined over the query concept e_i, and w_j's are the free parameters. Hence, λ_i depends on the query via g_j and w_j. Essentially, the ranking function is now *query-dependent* due to this parameterization. For the concept-importance features g_j, both collection features (collection frequency and document frequency), features from external sources (English Wikipedia and ClueWeb09), and a constant feature are used. They are summarized in Table 3.2. We note that separate sets of concept-importance features are defined for query unigrams concepts and bigrams concepts in the WSD

model, and they are denoted by g_j^t and g_j^b, respectively.

3.2 Basic setup

As a quick review, the basic setup for "learning to efficiently rank" is similar to the standard supervised "learning to rank" [2], where we would like to learn a ranking model from a training dataset to optimize a performance metric. The difference is in addition to using training queries and relevance judgements as inputs to learning, we also employ a cost model that estimates total run-time costs of different models, which helps us to derive cost-sensitive models during learning.

As always, the first step in learning any ranking model is to identify what the model optimizes (i.e., optimization metric). In this chapter, we first introduce a new class of optimization metrics for capturing the mean tradeoff between effectiveness and efficiency, called "Mean Effectiveness-Efficiency Tradeoff" (MEET) in Section 3.3. Then, we introduce a new class of ranking models (*efficient linear models*) built for directly optimizing this metric to achieve high quality retrieval results in reasonable amount of time in Section 3.4. We then present our training procedure in Section 3.5 and experiment results in Section 3.6.

3.3 Tradeoff metrics

In this section we introduce a class of tradeoff metrics between effectiveness and efficiency that our proposed efficient linear models will optimize. This class of metrics is tunable, which means they can adapt to different user requirements. We

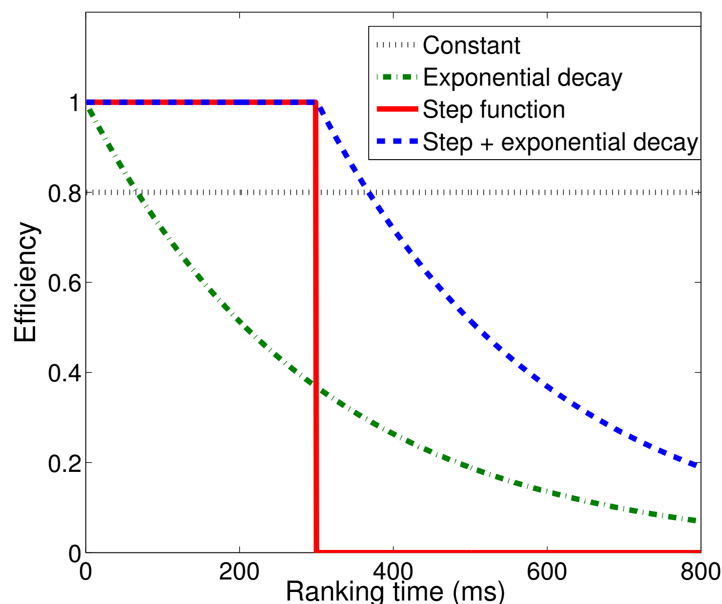


Figure 3.2: Efficiency functions. Constant and exponential decay are self explanatory. The step function and step + exponential function can model time preferences (threshold t=300ms here); when exceeding the time requirement, the ranking model either gets zero efficiency value or an exponentially lower efficiency value (resp.)

define a general class of efficiency functions and describe how different functions yield different tradeoffs. We begin by describing how to measure the efficiency and effectiveness of retrieval models.

3.3.1 Measuring efficiency

The efficiency of a search engine can be measured in many different ways, 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 proposal, 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 unbounded in theory. 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 $\sigma(\cdot) : \mathsf{R}^{+} \rightarrow [0, 1]$ 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:

$$\sigma(Q) = 1, \text{if } \tau(Q) \leq t$$

$$\sigma(Q) = 0, \text{if } \tau(Q) > t$$

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:

$$\sigma(Q) = 1, \text{if } \tau(Q) \leq t$$

$$\sigma(Q) = \exp(\alpha \cdot (\tau(Q) - t)), \text{if } \tau(Q) > t$$

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 3.2 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.

3.3.2 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 chapter 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.

3.3.3 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 $\gamma(Q)$:

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

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.

Discussions on tradeoff metrics:

The learning problem belongs to the family of multi-objective optimizations. Although there are many ways to combine multiple metrics to derive a final objective, the final objective should satisfy two basic requirements: 1) it needs to capture the tradeoffs between model speed and effectiveness, and provide a simple way to

control their relative importance in the final model; 2) the final objective must be optimizable by the selected learning to rank algorithm. We note that the second requirement is non-trivial, since it is a well-known fact that many classes of learning algorithms can not optimize for an arbitrary objective metric [1]. Thus, we must jointly select the appropriate learning algorithm with the final objective metric.

The line-search algorithm, employed for constructing efficient linear models, can actually optimize any arbitrary objective due to its non-analytical nature [15], which makes selecting an objective function relatively simpler than otherwise. Amongst the space of all choices for the final objective (e.g., linear functions, divisions, multiplications, harmonic means, etc.), we use harmonic means since they are widely accepted as a way for deriving a single metric from multiple competing measures in the IR literature. Furthermore, they provide a simple and tunable way for controlling the relative importance between model speed and effectiveness (via the weighting parameter β).

In contrast to line-search algorithms, we introduce an analytical boosting-based solution for constructing fast cascades in Chapter 5, which requires the final objective metric to be linear. Detailed discussions on the linear objective and a proof on why the objective works with the proposed boosting algorithm are given in Chapter 5.

3.4 Efficient linear models

In this section, we introduce a new class of ranking models which have a mechanism for balancing between retrieval effectiveness and efficiency. Since efficiency is inherently a query-time measure (e.g., efficiency cannot be measured by the set of offline selected features, since it depends on the actual queries), we need to augment the linear model with the ability to adapt to query-time efficiency. We achieve this by query-dependent feature pruning (i.e., removing query-dependent features that do not contribute much to effectiveness but incur high complexity). As an instance of the efficient linear model, we focus on improving the efficiency of the (weighted) sequential dependence model by selectively pruning features in a query-dependent manner. We call the resulting model *efficient sequential dependence model* (ESD). We note that both the feature weights and pruning threshold in the ESD model are learned to optimize a given efficiency/effectiveness tradeoff metric.

3.4.1 Limitations of linear models and offline feature selection

Before going into the details of our models, we first point out the limitations of the linear feature-based models and offline feature selection for balancing retrieval effectiveness and efficiency.

Linear ranking models that use a large number of features, such as those in learning to rank [2], can be highly effective. However, when applied to ranking tasks over large-scale document collections, they can result in long query execution times due to evaluating many query-dependent features at query-time. A natural

way to lower such high costs is to reduce the set of features used by these ranking models. Feature selection for ranking [62] has recently been proposed in the IR community, and it considers offline feature selection for ranking models. Although such offline feature selection can help to reduce the overall query evaluation costs (due to the reduction of feature set), it can not fully capture retrieval efficiency, because efficiency is inherently a query-time measure and different queries can have very different query evaluation times even if the *same* set of offline selected features are used for them (and the query-document features may need to be evaluated at query-time). An example for illustrating this claim is given in Table 3.3. There are two ranking features, where the first feature is a term frequency (TF) feature, which counts the occurrences of individual query term in each document, and the second feature is a bigram feature, which counts the occurrences of each query bigram in each document. When these same features are used for two different queries "White House" and "White House Rose Garden", we see that the query evaluation costs can be vastly different. The costs for evaluating the first query involve two term frequency feature evaluations, one for each query term, and one bigram feature evaluation for the entire query; whereas evaluating the second (longer) query is more costly, since it needs four term frequency evaluations, again one on each query term, and three bigram feature evaluations on the three bigrams contained in the query.

Another limitation is that for a given set of features, the linear model's parameter space (i.e., feature weights) can only be learned to account for retrieval effectiveness, and there is no a mechanism for handling efficiency within the model parameter space. To overcome both of these issues, in the next section we propose

Feature set	**Queries**	**Evaluation Costs**
{Term frequency, bigram}	White House	TF: 2 unigrams Phrase: 1 bigram
	White House Rose Garden	TF: 4 unigrams Phrase: 3 bigrams

Table 3.3: Limitations of linear models: offline selected features cannot capture query-time retrieval efficiency.

efficient linear models which utilize *query-dependent* feature pruning to improve efficiency while being cognizant of retrieval effectiveness.

3.4.2 Efficient linear models

In this section, we describe a class of linear ranking functions that 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. The key idea is to perform query-dependent feature pruning to improve efficiency, with minimal impact on effectiveness.

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

$$Score(Q, D) = \sum_i \lambda_i f_i(Q, D) \tag{3.2}$$

where Q is a query, D is a document, $f_i(Q, D)$ is a feature function, and λ_i is the weight assigned to feature i. Recall each weight λ_i also takes on a parametric form,

as follows:

$$\lambda_i(e_i) = \sum_j w_j g_j(e_i)$$

where e_i is the query concept that f_i is defined over, $g_j(e_i)$ is a concept-importance feature on e_i, and w_j is the weight assigned to the concept-importance feature. Notice since the weights λ are now *query dependent*, they can adapt better to different query scenarios via the feature functions g_j. We will show shortly that allowing λ to depend on Q provides an intuitive way to prune features.

We focus on a particular instance of a highly effective ranking function that takes on this functional form – the WSD model [34]. The definitions for the ranking features and the concept-importance features used by the WSD model are summarized in Table 3.1 and Table 3.2, respectively, in the background section 3.1.

Models of this form provide a natural mechanism for eliminating features in a query-dependent manner, thereby improving efficiency (due to the reduction of the number of query-document features to be evaluated at query-time). We propose to drop features according to the magnitude of $\lambda_i(e_i)$, the query-dependent weight assigned to feature i. If $\lambda_i(e_i)$ 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 $|\lambda_i(e_i)| \leq \epsilon$, 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 [20] 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:

$$\frac{\lambda_i(q_j, q_{j+1})}{\lambda_i(q_j) + \lambda_i(q_{j+1})} \leq \epsilon$$

where q_j q_{j+1} constitute a query bigram concept e_i. 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. We call the resulting model an *efficient sequential dependence* model (ESD). Preliminary experiments found the general strategy of pruning according to $|\lambda_i(e_i)| \leq \epsilon$ 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.

A main advantage of this simple query-dependent pruning technique is that it is very fast, which is a desirable property given that we need to eliminate query-dependent features and construct the actual ranking function at query time. As our experimental results show, this technique also results in highly effective ranking functions for a given tradeoff metric.

3.5 Parameter estimation

In this section we describe our method for automatically learning the parameters of our proposed model from training data. In addition to learning the

parameters w, we must also learn the concept pruning threshold ϵ. Although there are many learning to rank approaches for learning a linear ranking function (i.e., estimating w) such as SVM [40] and SVM^{MAP} [33], 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 in closed-form, 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 [41]. The algorithm iteratively optimizes the metric by performing a series of one-dimensional line searches in the MEET metric space. At each iteration, it searches for an optimal value for each parameter while holding all other parameters fixed. This iterative process continues until there is no further improvement in the objective metric. 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.

We use this algorithm 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

	Wt10g	Gov2	Clue
topics	451-550	701-850	1-50
# docs	1,692,096	25,205,179	50,220,423
avg qlen (title)	2.50	2.96	1.88
avg qlen (desc.)	6.08	5.90	5.88

Table 3.4: Number of docs, topics, and average query lengths of test collections.

for directly optimizing non-smooth functions using similar types of hill-climbing methods, such as simultaneous perturbation stochastic approximation [63], which uses fewer function evaluations and could be used to speed up training.

3.6 Experiment results

This section presents our experimental results. We begin by describing our experimental setup and then present a detailed evaluation of our proposed method using publicly available TREC datasets.

3.6.1 Experiment setup

We implemented our learning to efficiently rank framework on top of Ivory, an open-source web-scale information retrieval engine [64]. 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).

Our experiment framework [64] currently does not employ caching. In real-life search engines, many frequent user queries can be answered by a result cache without the need for scoring. Our query set can be viewed as being sampled from queries that cannot be answered by the result cache. This same assumption has also been used in prior work on search engine efficiency [6].

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 3.4. 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 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 index and score the top 1000 returned documents. We use Dirichlet-based subset of features in Table 3.1 for our experiments.

We compare the ESD model with two baseline models. One is the bag-of-words query likelihood model [12] (QL), with Dirichlet smoothing parameter $\mu = 1000$. The other is the less efficient, but more effective sequential dependence model [15] (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 optimize the MEET metric. A Wilcoxon signed rank test with $p < 0.05$ was used to determine the statistical significance of differences in the metrics.

3.6.2 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 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.

3.6.2.1 Tradeoff between effectiveness and efficiency

Table 3.5 presents results of two sets of experiments using the step + exponential function, with what we subjectively characterize as "slow" decay and "fast"

“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*	22.12	7.09	33.57*	32.91	13.46	21.68	21.41
ESD	0.235	24.04*†	23.03*†	6.42	34.74*†	33.94*†	11.18	22.34*	22.14

“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*	19.63*	7.09	33.57*	31.77	13.46	21.68	21.26
ESD	0.215	23.42*	21.55*†	5.46	33.65*	32.58	8.55	21.24	21.08

Table 3.5: 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.

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 3.5. 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 has similar or better effectiveness than SD. 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, in 3 of 6 cases ESD achieved statistically significantly higher MEET than QL, and that in the other three cases the apparent differences were not statistically 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.

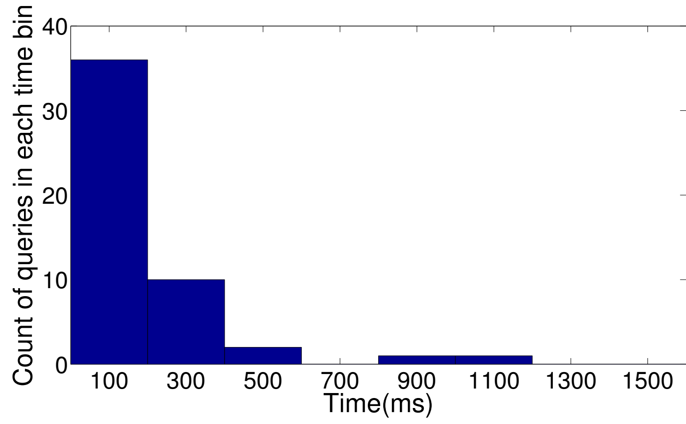
Lastly, note that setting the time target t in the efficiency function implies that the ranking model will be penalized by an exponential decay in efficiency if its query ranking time exceeds t. This is a soft penalization factor, which contrasts with the more harsh step function where efficiency is 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 3.5, where the average query time of ESD is consistently greater than the time threshold t.

3.6.2.2 Analysis of query latency distribution

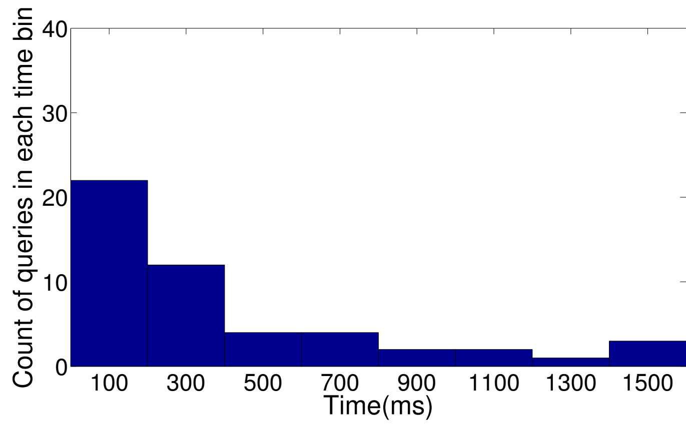
Another benefit of our proposed framework is that learned models exhibit low variance in query execution times. Figure 3.3 plots histograms of query execution for QL, SD, and ESD on the Wt10g collection. The ESD model was trained using the step + exponential (fast decay) efficiency function.

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.

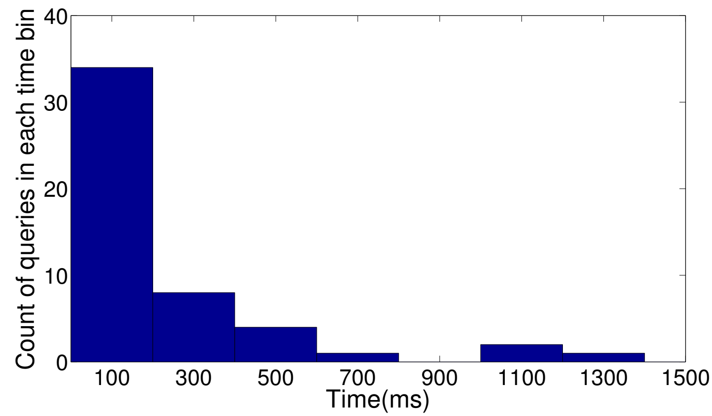
Why is reduced variance in query execution time important? For real-world search engines, it is important to ensure that a user, *on average*, gets good results quickly. However, it is equally important to ensure that *no user* waits too long, since these represent potentially dissatisfied users who never come back. A basic



(a)



(b)



(c)

Figure 3.3: Distribution of query execution time for Wt10g queries for (a) query likelihood (QL); (b) sequential dependence model (SD); (c) efficient sequential dependence model (ESD).

principle in human-computer interaction 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.

3.6.2.3 Relationship to other retrieval models

Finally, we demonstrate that previous ranking models that consider effectiveness 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

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

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 3.6 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 [34]. The ESD model in this case significantly outperforms SD in MAP and Meet scores; the differences are significant in two of the three collections.

Table 3.7 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

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

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 bigrams that have little impact on effectiveness, but nevertheless have an efficiency cost).

Chapter 4

Ranking under Temporal Constraints

In the previous chapter, we introduced the notion of *efficient linear models*, which are capable of balancing between retrieval effectiveness and efficiency according to a tradeoff metric. In this chapter, we introduce the notion of temporally constrained ranked retrieval, which produces the best possible ranked list for a given query under a time constraint. It is different from the previous chapter since now the time constraint is imposed on a *per-query* basis and the goal is to construct effective ranking models that can satisfy the imposed time constraints.

There are several practical motivations for ranking under temporal constraints. First, users are diverse and have different tolerances to query execution times. Some are impatient and want results as soon as possible, while others may be willing to wait a bit longer if it means better results. Second, there may be a service level agreement (SLA) specifying the expectations about the quality of service, such as query execution time, which the system should conform to. Third, the ability to impose constraint on query execution time will help the system to adapt to excessive query volume. For example, when query load is high, we might tighten the temporal constraints to maintain roughly the same query latency, at the cost of some reduction in quality—this might be preferable to forcing users to wait longer for results.

Naturally, the quality of results improves as more computation time is allowed,

but critically, the ranking function should always produce *some* results given an arbitrary time constraint. This property is desirable from a number of perspectives: to cope with diverse users and information needs, as well as to better manage system load and variance in query execution times. This idea is related to *anytime algorithms*, which were introduced in the mid-1980s by Dean and Boddy [65, 66] in the context of time-dependent planning. The primary difference, however, is that anytime algorithms provide a solution at any arbitrary point, whereas our temporally constrained ranking functions require the time constraint to be specified *in advance* (and may not produce *any* result prior to this constraint).

More formally, we define a temporally constrained ranking function as one that solves the following ranking task: given a user query q and a time constraint $T(q)$, the goal is to produce a top k ranking of documents from collection C that maximizes a metric of interest such as mean average precision, NDCG, etc.

It is true that we can address the diverse users and information needs by devising multiple ranking algorithms that encode specific tradeoffs, and then dynamically select the appropriate algorithm at query time. The obvious downside, however, is duplicate development effort. Instead, it would be more desirable to have a single ranking algorithm that comes with an adjustable "knob" to set the desired query evaluation time.

In this chapter, we propose two temporally constrained ranking algorithms based on a class of probabilistic prediction models that can naturally incorporate efficiency constraints: one that makes *independent* feature selection decisions, and the other that makes *joint* feature selection decisions. Experiments on three different

test collections show that both ranking algorithms are able to satisfy imposed time constraints, although the joint model outperforms the independent model in being able to deliver more effective results, especially under tight time constraints, due to its ability to capture feature dependencies.

4.1 Ranking with time constraints

This section formalizes the problem of temporally constrained ranking, where in addition to a query q, we specify a time constraint $T(q)$ for the query. We first provide a brief overview of the linear ranking functions and the feature set we use, and introduce the concept of temporally constrained linear ranking functions. We then present novel formulations based on probabilistic prediction models for constructing temporally constrained functions. Finally, we discuss how to satisfy the time requirement and estimate model parameters.

4.1.1 Linear ranking functions

As a quick recap from Section 3.1, a linear ranking function is characterized by a set of features $F = f_1, \ldots, f_N$ and the corresponding model parameters $\Lambda = \lambda_1, \ldots, \lambda_N$. Each feature f_i is a function that maps a query-document pair (q, d) to a real value.

We now describe the features f_i and g_j that we consider in this chapter. For the query-dependent concept-importance features g_j, we use the same set of features as in the previous chapter (Table 3.2), including collection-based (collection frequency

and document frequency), features from external sources (English Wikipedia and a large Web collection), and a constant feature. As for the ranking features f_i, both term-based features [12, 13] and term proximity features [21, 15] have been widely used in such models, and have been shown to be especially successful when used in combination [67]. We take a similar feature-oriented approach and use both term features and term proximity features as part of our feature pool (Table 3.1). Among them, we use two different unigram term features, each using a different feature scoring function (BM25 or Dirichlet). We use a set of term proximity features defined over bigrams within the query, where each is computed from a BM25 or Dirichlet scoring function, for a specific window type (ordered/unordered) and window length. These are similar to the features previously explored [67].

Note that as a result from this query-dependent weighting [34, 20], features f_i that are defined on the same unigram/bigram (i.e., query concept) in the query will have the same feature weight value λ_i. This property will be used for developing efficient algorithms for creating temporally constrained ranking functions in Section 4.2.1.

Finally, most of the computational cost associated with such ranking functions comes from the cost incurred from evaluating the features f_i. Unlike the features f_i, we assume that there is negligible cost associated with computing the features g_j. Typically, in an operational setting retrieval engines would have access to large-scale, low-latency distributed caches for these types of global statistics.

4.2 Constrained linear ranking functions

In a temporally constrained setting, using all features in the linear ranking function may exceed the time requirement, since the query-document features may need to be evaluated at query-time. Instead, for each query, we need to alter the efficiency characteristics of the ranking function by only using a subset of the features to meet its time requirement. We call the resulting ranking function a *temporally constrained* linear ranking function and it has the following general form:

$$
\begin{aligned}
Score(q,d) &= \sum_{f_i(q,\cdot):S_i=1} \lambda_i f_i(q,d) \\
s.t. \sum_{f_i(q,\cdot):S_i=1} & C(f_i(q,\cdot)) \leq T(q)
\end{aligned}
$$

where $C(f_i(q,\cdot))$ denotes the computational cost of evaluating feature f_i for q over the document collection, $T(q)$ is the time requirement for query q, S_i is a binary value denoting whether $f_i(q,\cdot)$ is computed for q, and λ_i's are the linear feature weights, parameterized by concept-importance features described in the previous section.

To instantiate a model, we must address the following: 1) how to best select the subset of query dependent features to construct the corresponding temporally constrained ranking function for each query q; 2) how to define the cost function $C(f_i(q,\cdot))$ for the linear features $f_i(q,\cdot)$ such that the response time of the resulting ranking function will not exceed the time requirement; 3) how to determine the free parameters (i.e., concept-importance feature weights w_j) for the temporally constrained ranking function. We described our proposed solution to these issues in

the following sections.

4.2.1 Prediction models

This section introduces two methods for creating temporally constrained ranking functions. Both are based on a class of probabilistic prediction models that can naturally impose time constraints on query execution times. The first method, which we call "Indep", makes independent decisions when selecting which features f_i should be evaluated. The second method, which we call "Joint", goes an extra step by accounting for query-level feature redundancy. We note that in contrast to previously proposed feature selection methods for ranking [62, 67], our problem is unique in that the feature selection is driven by the time constraint for each query, not purely by effectiveness.

In both methods, the decision on what features to use largely depends on the feature weights $\lambda_i(q)$. Features with large weights are more likely to be selected when facing a time constraint because they have a more significant impact on the final ranking. In particular, in the "Indep" model, the likelihood of a feature $f_i(q, \cdot)$ being selected is directly proportional to its query dependent feature weight $\lambda_i(q)$. In the "Joint" model, as we will show, the selection depends both on the feature weight and the feature's redundancy relationship with respect to other features.

4.2.1.1 Independent Prediction Model

The independent prediction model is based on a logistic regression model. It aims to directly estimate the likelihood that a given feature f_i should be included in the ranking function or not. Using this model, the probability of selecting feature f_i (i.e., S_i=1) is computed as:

$$P(S_i|q) = \frac{\exp(\lambda_i(q))}{1+\exp(\lambda_i(q))} = \frac{\exp\left(\sum_j w_j g_j(q,i)\right)}{1+\exp\left(\sum_j w_j g_j(q,i)\right)} \tag{4.1}$$

where the g_j are the concept-importance features used in our linear ranking function and the w_j are the corresponding model parameters. When defined in this way, the likelihood of choosing feature i is monotonic with respect to $\lambda_i(q)$, the query-dependent weight from our linear ranking function. As mentioned earlier, this is intuitively appealing, as it is desirable to choose features with large weights, since they are more likely to have more of a positive impact on the ranking.

Under this independent model, the joint probability of selecting features is simply the product of individual selection likelihoods. Given a time constraint $T(q)$ for query q, we need to infer the most likely selections over all features and construct the corresponding constrained ranking function $R(q)$, such that the time cost of $R(q)$ will be within $T(q)$. While many existing methods are available for general inference for prediction models [68], such methods are unconstrained. Instead, inference under time constraint $T(q)$ for a given query, can be easily shown to be equivalent to the following optimization problem:

$$\hat{S} = max_S \sum_i S_i \lambda_i(q) \tag{4.2}$$

$$s.t. \sum_i S_i C(f_i(q,\cdot)) \leq T(q), \qquad S_i \in \{0,1\}$$

where S_i is a binary variable indicating whether feature f_i and its corresponding feature weight λ_i will be used in the ranking function, $C(f_i(q,\cdot))$ is the cost of evaluating f_i for q over the collection, and $T(q)$ is the time constraint. The goal of the optimization is to find an optimal set of features that maximizes the objective and satisfies the time constraint $T(q)$. Similar approaches for casting inference as an optimization task have also been used in the NLP task of semantic role labeling [69]. This formulation has the advantage that it enables us to impose arbitrary linear constraints (such as temporal constraints) on the model outputs (i.e., the temporally constrained ranking function).

Solving the stated optimization problem in Equation 4.2 is done at runtime, so for each query, the complexity cannot be very high. Nonetheless, the Indep model allows for an extremely efficient inference from which the temporally constrained ranking function is constructed. Note that our query cost model excludes the inference cost because, as we will show, the running time complexity for constructing ranking functions is negligible for reasonably sized queries.

The process of inferring the temporal ranking function from the Indep model is presented in Algorithm 1. It should be easy to see that for Indep prediction model, its optimization problem corresponds to the classical *knapsack problem*. Here, features are "items" that we are trying to fit into the knapsack. The value of each corresponds to the feature weight $\lambda_i(q)$, the cost corresponds to $C(f_i(q,\cdot))$, which is explained in detail in Section 4.2.2, and the knapsack capacity is $T(q)$. We first

Algorithm 1: Independent Ranking

Input: Query time requirement T(q); ranking features $f_i(q, \cdot)$; concept-importance features $g_j(q)$ and concept-importance features weights w_j

Output: Temporal constrained ranking function R(q)

Compute feature weights: $\lambda_i(q) = \sum_j w_j \; g_j(q)$;

Compute feature profit density: $p_i = \frac{\lambda_i(q)}{C(f_i)}$;

Queue $\mathcal{F}$: features sorted in non-increasing order of profit density;

Initialize $R(q) = \{\}$;

Initialize $totalCost = 0$;

while *size*($\mathcal{F}$) > 0 **do**

  Remove f_i from head of $\mathcal{F}$;

  if $totalCost$ + cost(f_i) $< T(q)$ **then**

    add $\{f_i, \lambda_i\}$ to ranking function $R(q)$;

    $totalCost = totalCost$ + cost(f_i);

end

return $R(q)$;

compute the profit density for each feature f_i, which is defined by the ratio between its value and cost (i.e., $\frac{\lambda_i(q)}{C(f_i)}$). We then add features according to this density (largest first), until we can no longer add any more features without overshooting our time constraint, or until when we run out of features to use.

Since the number of features is linear in $|q|$, the length of the query, the time complexity for this process is $O(|q| \log |q|)$, which accounts for sorting and adding features into the constrained ranking function. This is trivial compared to the time taken for query evaluation.

4.2.1.2 Joint Prediction Model

The independent prediction model assumes that feature selection decisions are made independently of each other. While this assumption is reasonable for small feature sets, it may not hold as well for larger feature sets. Therefore, we would like to model relationships between features and eliminate the computation of redundant features that do not add very much to the ranking function in terms of relevance, yet result in increased query execution times.

We define redundancy only for features defined over the same query concept (i.e., unigram or bigram). The intuition is that under a time constraint, we would like to use features that provide maximal coverage over all query concepts, rather than repeatedly using features for the same query concept. Hence, for a pair of features f_i and f_j that share a common query concept (i.e., $\lambda_i(q) = \lambda_j(q)$), the redundancy is defined as follows:

$$r(f_i, f_j) = \begin{cases} 1 & \text{if } \lambda_i < \alpha \\ 0 & \text{otherwise} \end{cases}$$

where α is a weight threshold. This definition says that f_i and f_j are redundant if the feature weight does not exceed threshold α. This definition allows us to impose redundancy penalty on features defined over less important query concepts, as determined by the feature weight value.

We propose using an undirected probabilistic graphical model [70] to capture the feature importance and redundancy relationship between features when making the selection under time constraint. An undirected graphical model is a probabilistic

Algorithm 2: Joint Ranking

Input: Query time requirement T(q); ranking features $f_i(q, \cdot)$; concept-importance features $g_j(q)$ and concept-importance features weights w_j

Output: Temporal constrained ranking function R(q)

Compute feature weights: $\lambda_i(q) = \sum_j w_j \ g_j(q)$;

Compute feature profit density: $p_i = \frac{\lambda_i(q)}{C(f_i)}$;

Queue $\mathcal{F}_1$: features sorted in non-increasing order of profit density;

Initialize queue $\mathcal{F}_2$={} ;

Initialize $R(q) = \{\}$;

Initialize $totalCost = 0$;

For each concept e, let G_e = set of features defined on e;

Let λ_e denote the weight of features in G_e;

Let $covered[e]$=false for all e;

while *size($\mathcal{F}_1$) > 0 or size($\mathcal{F}_2$) > 0* **do**

  Remove f_i that has max p_i from head of $\mathcal{F}_1$ or $\mathcal{F}_2$;

  if $totalCost$ + cost(f_i) < $T(q)$ **then**

    add $\{f_i, \lambda_i\}$ to ranking function $R(q)$;

    $totalCost = totalCost$ + cost(f_i);

    Denote concept covered by f_i by $e\prime$;

    if ($covered[e\prime]$ is false and $\lambda_{e\prime} < \alpha$) **then**

      $\lambda_{e\prime} = \lambda_{e\prime}$ - β;

      $G_{e\prime} = G_{e\prime} \setminus f_i$; move $G_{e\prime}$ from $\mathcal{F}_1$ to end of $\mathcal{F}_2$;

      $covered[e\prime]$ = true;

end

return $R(q)$;

model defined over an undirected graph, which encodes the conditional independence assumptions amongst random variables, corresponding to the nodes in the graph. Here, we make use of the Ising models (a.k.a. Boltzmann machine) [71] to represent individual features and their pairwise redundancy relationships. Under this model, the joint probability over all feature decisions is:

$$P(S_1, \ldots, S_k) = \frac{1}{Z} \exp\left(\sum_i \left(\sum_j w_j g_j(q,i) S_i + \sum_{i' \in N(i)} \beta \cdot r(f_i, f_{i'}) S_i S_{i'}\right)\right) \tag{4.3}$$

where $N(i)$ are the features that share an edge with feature i in the graph (i.e., features in $N(i)$ are defined over same query concept as f_i), $r(f_i, f_{i'})$ is the redundancy feature defined over the pair $(f_i, f_{i'})$ and β is the model parameter associated with redundancy feature r. S_i, w_j and g_j are as defined earlier.

It can be easily shown that the probability of selecting features $f_1, \ldots, f_k$ under time constraint can be stated by the following optimization problem:

$$\hat{S} = max_S \sum_i \lambda_i(q) S_i + \sum_i \sum_{i' \in N(i)} \beta \cdot r(f_i, f_{i'}) S_i S_{i'}$$
$$s.t. \sum_i S_i C(f_i(q, \cdot)) \leq T(q), \quad S_i \in \{0, 1\}$$

The goal of the optimization is to find a set of features that reach an optimal balance between feature effectiveness and redundancy, while satisfying the time constraint $T(q)$. Note that when $\beta = 0$ this problem reduces to the independent case.

The optimization problem is an integer linear program [72], and several practical solutions exist. Most commonly, they iteratively make decisions based on a score value for each feature (similar to the knapsack problem). However, they require continually re-sorting the features after a feature is added and the values of

its neighbor features are re-computed to account for redundancy. Our problem has the nice properties that all features defined over the same query concept share a common feature weight value, and their redundancy feature is binary valued. These properties enable us to derive an efficient solution (Algorithm 2) for this optimization problem, and as we will show, it has the same running time complexity as Algorithm 1, but however, it significantly improves effectiveness over Indep model as shown by our experiment results.

Similar to the independent case, we add features according to their profit densities (largest first). When a feature f_i is added, if its feature weight does not exceed α, the selection likelihoods of its remaining neighboring features are recomputed to account for feature redundancy penalty (if not already done). In this case, feature ordering needs to be adjusted to account for new weights. But because features defined on the same query concept share same weight value, there is no need to sort them after they get same penalty β. To make sure they are properly placed with respect to other features, we maintain two queues, $\mathcal{F}_1$ contains non-penalized features as sorted originally, and $\mathcal{F}_2$ contains the penalized features, which are always sorted as explained earlier. This way, there is no need to repeatedly perform re-ordering operations after new weights are computed. We select the feature with the best profit density between $\mathcal{F}_1$ and $\mathcal{F}_2$ as the next feature for the temporally constrained ranking function during each iteration.

The time complexity for this process is $O(|q| \log |q|)$, accounting for the initial sorting of features and for creating the ranking function. In practice, this cost is trivial for reasonably sized queries.

4.2.2 Temporal constraint enforcement

In this section, we describe how we define $C(f_i(q, \cdot))$, the cost for evaluating feature $f_i(q, \cdot)$ for q over the document collection. More specifically, we are interested in characterizing the efficiency of a linear ranking function as a result of using a particular subset of features. Efficiency can be interpreted in two ways: 1) in an absolute sense, in terms of the total amount of time necessary to compute the ranked list, or 2) in a relative sense, where given a baseline ranking model, we measure how much more (or less) efficient our proposed method is compared to the baseline. We adopt the second interpretation because it factors out differences in hardware.

Thus, we model the query execution time relative to some baseline ranking function. This type of cost model requires us to capture the costs associated with evaluating different features f_i within the linear ranking function. The cost for a linear ranking function $R(q)$ is computed as the sum of its individual feature costs:

$$C(R(q)) = \sum_{i=1}^{N} C(f_i(q, \cdot))$$

We estimate $C(f_i(q, \cdot))$ as the sum of document frequencies $DF(t)$ of each term t required to compute f_i. That is,

$$C(f_i(q, \cdot)) = \sum_{t \in f_i} DF(t)$$

For features defined over unigrams, this sum has a single component; while for features defined over bigrams, it has two components. Intuitively, this analytical model captures the fact that evaluating more features and evaluating each feature over long posting lists (i.e., large DF values) will both result in greater time complexity.

Given a baseline ranking function, say we want to construct an temporally constrained ranking function $R(q)$ such that the actual execution time of the model must be within a multiple k of the baseline model's query execution time $R_b(q)$. This can be expressed as the following constraint:

$$C(R(q)) \leq k\ C(R_b(q))$$

where the cost is computed as described above. Our algorithm for constructing $R(q)$ enforces this constraint. As we show in the experiment section, this very simple cost model works surprisingly well for ensuring the actual time $T(R(q))$ of the model meets the *actual* time constraint imposed on the query:

$$T(R(q)) \leq k\ T(R_b(q))$$

and $k\ T(R_b(q))$ is the time requirement for q, as a multiple of baseline time $T(R_b(q))$.

4.2.3 Parameter estimation

It should be clear that a new optimization metric must be defined here, because commonly used effectiveness metrics (MAP, P20, etc.) do not directly account for how the effectiveness of models vary across a range of time constraints.

To evaluate the performance of a temporally constrained ranking function, the new metric must account for the expected effectiveness over different time budgets. Stated more formally, the expected effectiveness $\mathcal{E}_q$ for a query q is defined as:

$$\mathcal{E}_q = \sum_t e_q(t)P(t)$$

where $e_q(t)$ denotes the effectiveness achieved by the constrained ranking function associated with time constraint t, and $P(t)$ represents the likelihood of t being assigned as the time constraint for the query. Any commonly used effectiveness metrics (such as average precision, P20, etc.) can be used in the effectiveness measure $e_q(t)$ at each single time point t. In our experiment section, we report results from using average precision, and P20.

For simplicity, we assume all t values are equally likely (i.e., uniform distribution for $P(t)$). However, we can assign various other distributions (i.e., Gaussian, exponential, etc.) to model more complex distributions for time constraints.

Taking the mean of $\mathcal{E}$ over a set of n queries gives us the mean expected effectiveness:

$$\mathcal{M}_E = \frac{1}{n} \sum_q \mathcal{E}_q$$

which represents the overall performance of the ranking functions across different time constraints and queries.

The set of free parameters we need to learn offline are the concept-importance feature weights w_j and the redundancy feature weight β and threshold α (for Joint model only). The values of these parameters versus the objective metric $\mathcal{M}_E$ are not smooth and there are multiple local maxima, so the standard gradient-search based methods can not be applied directly. In this work, we employ a simple line search algorithm for optimizing the various model parameters by directly optimizing the mean expected effectiveness $\mathcal{M}_E$ on the training set. This approach has been used by several earlier work for estimating parameters in ranking functions to directly to

optimize objective metrics [34, 15]. The algorithm iteratively optimizes the metric by performing a series of one-dimensional line searches. At each iteration, it searches for an optimal value for each parameter while holding all other parameters fixed. This iterative process continues until the improvement in the objective metric drops below a threshold. More details of this parameter estimation can be found in [72].

4.3 Experiments

We report experimental evaluation of our proposed temporally constrained ranking models on three TREC web test collections: Wt10g, Gov2, and Clue (first English segment of ClueWeb09). Details of these test collections are provided in Table 3.4 in Chapter 3. The title and description portions of the corresponding TREC topics were used as queries, split equally into a training and test set. All parameter tuning was performed on the training set, and all of the results reported are from applying the learned parameters to the test set.

Our experiments compare average precision (AP), precision at 20 (P20), and mean expected effectiveness $\mathcal{M}_E$ of various models, including a baseline query-likelihood (QL) model, a baseline sequential dependence (SD) model [15], the Indep model, and the Joint model. In addition, we constructed an upper-bound model, called "All features", which is a standard learning to rank model (i.e., optimized for effectiveness only with no temporal constraints) learned over the entire feature set. To remain consistent with our proposed models, the "All features" model is constructed in the same manner as the temporally constrained models, except the

	Wt10g	**Gov2**	**Clue**
Title	0.18s	2.2s	4.2s
Description	0.48s	7.7s	20.5s

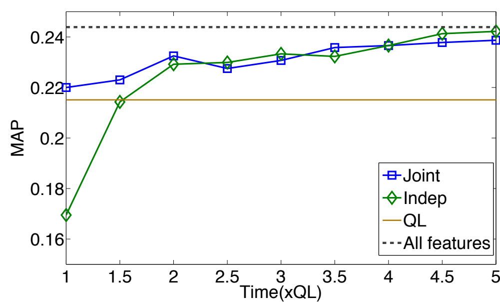
Table 4.1: Avg. query execution time of baseline QL for title and desc. queries of Wt10g, Gov2, Clue.

temporal constraint is set to infinity. The Wilcoxon signed rank test with $p < 0.05$ was used to test for statistically significant differences between the methods.

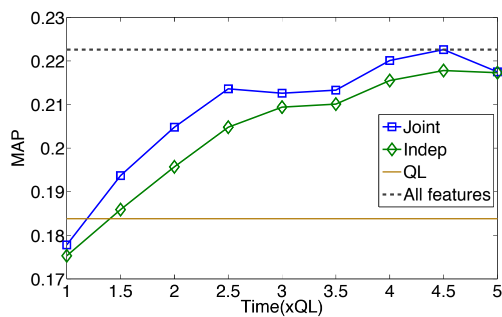
We implemented our learning to efficiently rank framework on top of Ivory, a newly-developed open-source web-scale information retrieval engine [64]. All experiments were run on a SunFire X4100, with two Dual Core AMD Opteron Processor 285 at 2.6GHz and 16GB RAM (although all experiments used only a single thread). Recall that we varied the time constraint for each query q be a multiple k of the query execution time of baseline QL, denoted by $k\ T_{QL}(q)$. The average query execution times of baseline QL for various data collections and queries are shown in Table 4.1 for reference.

4.3.1 Effectiveness vs time constraints

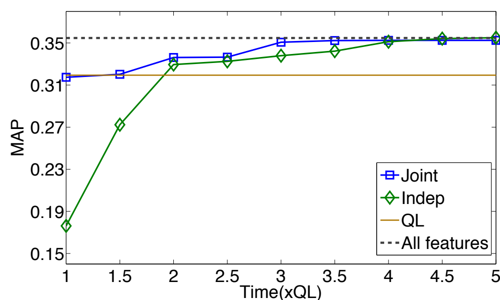
Figure 4.1 compares the MAP of our proposed temporally constrained ranking algorithms as a function of time constraints, from $1T_{QL}(q)$ and $5T_{QL}(q)$ in increments of $0.5T_{QL}(q)$ for each query. In each graph, the effectiveness of the baseline QL model is plotted as the lower solid line, and the "All features" upper-bound is plotted as the upper dotted line.



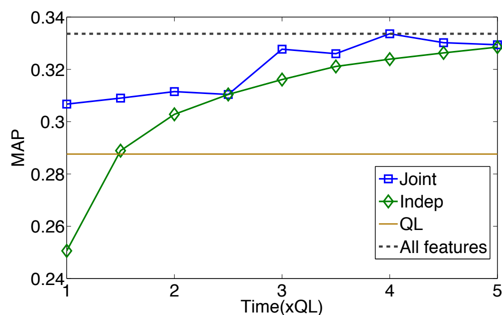
(i) Wt10g title



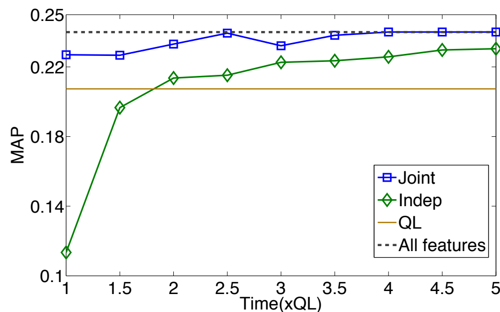
(ii) Wt10g description



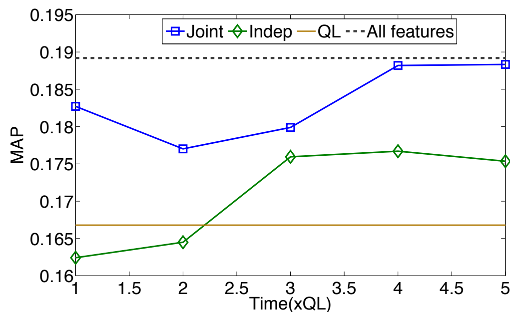
(iii) Gov2 title



(iv) Gov2 description



(v) Clue title



(vi) Clue description

Figure 4.1: MAP versus time for Indep, Joint and QL models on title and description queries in the test sets of Wt10g, Gov2 and Clue.

While in general both the Indep and Joint models produce more effective results than the baseline QL when given more time, the Joint model consistently achieves equal or higher MAP than the Indep model across all time constraints for both title and description queries. It also approaches the upper bound more rapidly as the time constraint is relaxed. This shows that the Joint model is superior to the Indep model across a wide range of retrieval scenarios.

Another interesting aspect is that the effectiveness difference between the Indep and Joint models is largest under tight temporal constraints, while the two models tend to perform similarly when the temporal constraint is relaxed. This can be explained by the fact that eliminating redundant features is more critical under strict temporal constraints—in such a case, using a diverse set of features is preferred. The effect of redundant features diminishes as the time constraint increases, since there is sufficient time to evaluate more features (redundant or otherwise). Feature redundancy also explains why the Indep model is different with respect to Joint and QL for very strict temporal constraints: it repeatedly choose features based on profit density alone. Thus, the time constraint will likely be violated before (at least) one feature for each facet in the query can be selected. Note this is not a problem with baseline QL, which only uses one feature for each term so there are no redundant features. In other words, given a rich feature set, under a strict time constraint, effectiveness is more critically dependent on selecting both high quality and non-redundant features.

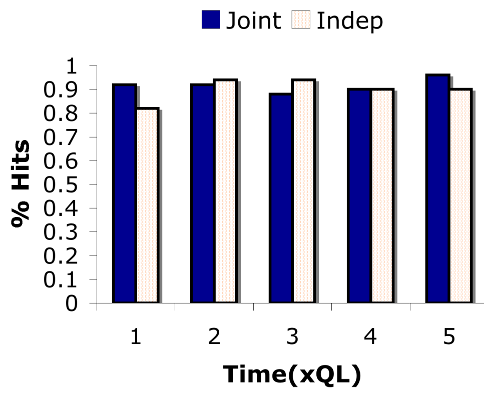
In general, MAP increases as the time constraint is loosened, for both the Joint and Indep models. Once again, this makes sense since the models are able to benefit

from more features. However, we encounter a point of diminishing returns around 4 T_{QL}, which suggests that using a very large number of features, as considered by previous effectiveness-centric ranking models, may not be very desirable in our formulation of temporally constrained ranking, at least on these datasets. In a learning-to-rank scenario that only takes effectiveness into account (which might roughly correspond to the right edges of our graph), we feel that the marginal gains in effectiveness that comes with evaluating more features (thus taking more time) trades off too much efficiency for a small effectiveness gain. This speaks to the advantage of our temporally constrained ranking functions, which allows the tradeoff between effectiveness and efficiency to be explicitly considered.

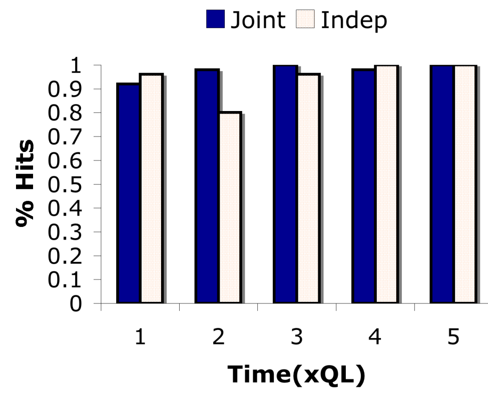
4.3.2 Satisfying time constraints

Recall that our constrained ranking functions use an analytical cost model based on the sum of document frequencies, which are highly correlated to actual query evaluation times. However, whether the time constraints are *actually* met on the test data is an empirical question, since there are no theoretical guarantees. As described in Section 4.2.1, the time for constructing ranking functions is negligible compared to retrieval time for reasonably sized queries, thus it is not included in the query evaluation times.

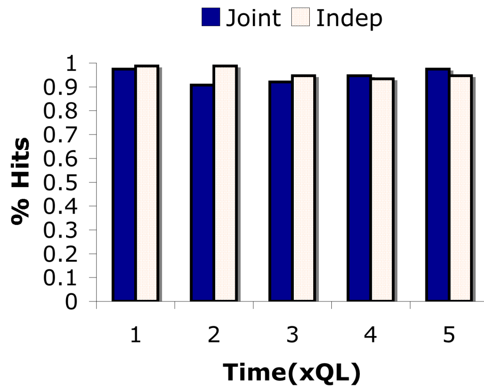
Figure 4.2 illustrates how well, in reality, the temporally constrained ranking functions satisfy query-specific time constraints on the test data. Each set of bars represents a particular time constraint, defined as before in terms of multiples of



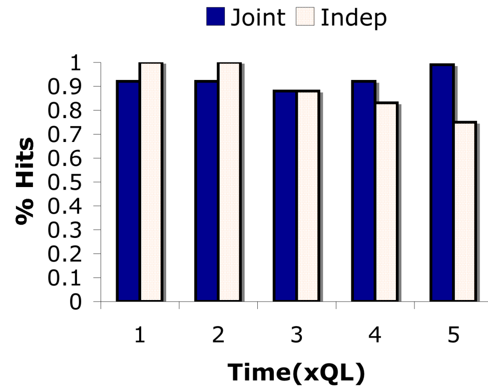
(i) Wt10g title



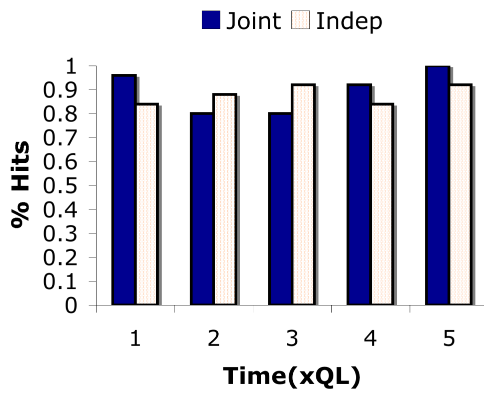
(ii) Wt10g description



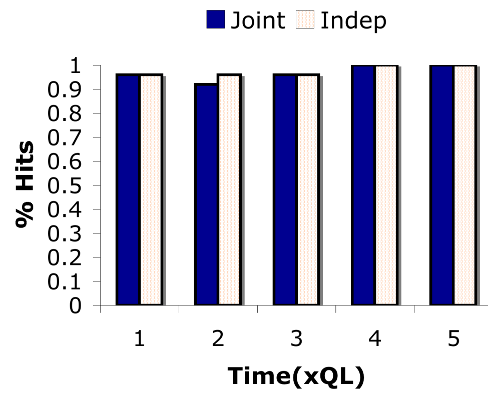
(iii) Gov2 title



(iv) Gov2 description



(iv) Clue title



(v) Clue description

Figure 4.2: The fraction of query evaluation times that satisfy the imposed time constraint for title and description queries of Wt10g, Gov2, and Clue datasets.

the baseline QL query evaluation time. The height of each bar shows the fraction of queries that satisfy the required time constraint. For each time constraint t, the temporally constrained ranking function is said to have satisfied the constraint if the actual query execution time is less than or equal to t. We see that the "hit rates" (i.e., meeting the specified time constraint) for both ranking algorithms are well above 80% for almost all test collections, while most of the values are above 90%. This suggests our sum-of-*dfs* cost model, which is used by the algorithm to guarantee the time constraints, works reasonably well in practice.

What is not shown on the graphs is that the same queries were missed across multiple time constraints. For instance, the query "maps of United States" missed all of the time constraints for the Clue collection. We found the queries that often missed their targets usually contained many frequent terms and/or are longer than average. This suggests that our analytical model underestimates the cost of these 'outlier' queries. As part of future work, we would like to explore ways to improve our analytical model to better handle such queries to minimize the number of queries that miss the specified time constraint.

Finally, a query is classified as a "miss" if its response time is greater than the required time t. However, if it only exceeds t by a very small amount, it may still be tolerable in practice. For instance, by allowing the response time to exceed t by a small amount (e.g., 5%), we observed that our models achieved 100% hit rates in many cases.

4.3.3 Expected effectiveness across constraints

To investigate the average effectiveness of our proposed temporally constrained ranking algorithms across different time points, we compared their mean expected effectiveness (according to MAP and P20) in Table 4.2 for title and description queries across all three test collections. Results from the QL baseline are also reported. In terms of averaged effectiveness computed using MAP, the Joint model consistently and significantly outperforms both Indep and QL. Specifically, for title queries, the Joint model attains over $7.5\%, 6.8\%$, and 13.3% improvements over QL on Wt10g, Gov2, and Clue, respectively. The Joint model achieves over $2.7\%, 7.7\%$, and 13.1% over Indep model on the same collections, respectively. Similar improvements are observed under averaged effectiveness computed using P20. We also note that the improvements of Joint over QL and Joint over Indep are statistically significant in two out of three collections, under both of the effectiveness metrics. The Indep model improves over QL in two out of three collections, under both effectiveness metrics, although the improvements are not statistically significant for title queries.

For the description queries, both of the Indep and Joint models achieve significant improvements over QL across all collections, while the Joint model also outperforms the Indep model in all three collections. The improvements of the Indep model over QL in terms of average effectiveness using MAP are $10.7\%, 7.0\%$, and 1.5%, respectively on the three collections. In two of these three cases the improvements of Indep over QL are statistically significant. Similar results are obtained under averaged effectiveness computed from P20. Compared to title queries,

Title Queries

	Wt10g			Gov2			Clue		
	M_E(MAP)	M_E(P20)	T_{98}	M_E(MAP)	M_E(P20)	T_{98}	M_E(MAP)	M_E(P20)	T_{98}
QL	21.51	32.40	–	31.94	50.93	–	20.74	37.25	–
Indep	22.54	32.77	4.5	31.67	50.37	4	20.78	35.01	–
Joint	**23.14***	**33.22**	4.5	**34.12**$^{*}_{\dagger}$	**54.76**$^{*}_{\dagger}$	3	**23.51**$_{\dagger}$	**40.54**$^{*}_{\dagger}$	2.5

Description Queries

	Wt10g			Gov2			Clue		
	M_E(MAP)	M_E(P20)	T_{98}	M_E(MAP)	M_E(P20)	T_{98}	M_E(MAP)	M_E(P20)	T_{98}
QL	18.38	28.60	–	28.76	49.47	–	16.68	29.90	–
Indep	20.35*	31.18*	–	30.76*	51.25	5	16.93	34.69	–
Joint	**20.84**$^{*}_{\dagger}$	**31.45***	4	**32.05**$^{*}_{\dagger}$	**52.82***	3	**18.18**	**37.05**	4

Table 4.2: Mean average effectiveness in terms of averaged MAP and averaged P20 from times $1T_{QL}$ to $5T_{QL}$ for title and description queries in the test sets of Wt10g, Gov2, and Clue. Bolded values denote best performance obtained for test queries in each data set. The $*$ and $\dagger$ symbols represent statistically significant differences with respect to QL and Indep, respectively. Percentage improvement shown in parentheses: over QL for Indep, and over QL/Indep for Joint.

this demonstrates that selecting features according to query dependent weights under time constraints (i.e., largest weights first), as employed by the Indep model, is more beneficial for verbose queries than short title queries. This is in line with previous findings [20, 34] for creating effective-centric ranking models (i.e., no time constraint), where query dependent weighting was shown to be more useful for long queries. The Joint model consistently outperform both QL and Indep by large margins for the description queries, and in most cases the differences are significant, which confirms our earlier observation that by reducing feature redundancy and selecting high quality features, the joint model can achieve high effectiveness across a wide range of time constraints. Note that for description queries, the Joint model outperforms the Indep model even on looser time constraints because there exists more redundancy in longer queries.

In addition to effectiveness, we are also interested in measuring how quickly each model converges to the effectiveness of the "All features" upper-bound. To approximate this measure, we let T_d denote the time (relative to T_{QL}) taken by each model to achieve $d\%$ of the upper-bound effectiveness, where $d\%$ is chosen to be 98% in our experiments, where effectiveness is measured in MAP. In terms of T_{98}, we see that for both title and description queries the Joint model has quantitatively better convergence rates than the Indep model across all collections. Furthermore, unlike Indep model, Joint model is able to attain the 98% of upper-bound effectiveness across all collections and query types. We note that the QL baseline is not able to attain the specified effectiveness (i.e., 98% of upper-bound effectiveness), thus it is not meaningful to compute T_{98} for it.

	Title Queries			**Description Queries**		
	Wt10g	Gov2	Clue	Wt10g	Gov2	Clue
SD	22.44	33.57	21.68	19.78	29.84	18.36
Indep	**23.66***	35.12*	22.57	21.55	32.39*	17.67
Joint	23.66	**35.25***	**24.00**	**22.01**$_{\dagger}$	**33.36***	**18.89**

Table 4.3: MAP scores of SD, Indep and Joint at T_{SD} for title queries (left) and description queries (right) for Wt10g, Gov2, and Clue. The $*$ and † represents sig. difference with respect to SD and Indep, respectively.

4.3.4 Comparison with SD model

How does our temporally constrained ranking models compare with a previously proposed effectiveness-centric model? The sequential dependence model (SD) [15] is a widely-used term proximity retrieval model, which adheres to a rigid efficiency-effectiveness tradeoff (i.e., it does not adapt to time constraints), and has demonstrated good performance across various tasks [15, 20, 34]. For fairness of comparison, we calibrate the time constraints of our temporally constrained models with respect to the SD model's time T_{SD}, where T_{SD} is approximately equal to $4T_{QL}(q)$ for each query q. Table 4.3 reports the results on title and description queries for the SD, Indep, and Joint models.

From these results, it is evident that both the Indep and Joint models significantly outperform SD when under the same temporal constraint as the SD model.

For title queries, the gains in MAP range between 4.1% and 5.4% for Indep, and range between 5.0% and 10.7% for the Joint model. Further, we observe that the gains of Indep over SD are greater for description queries than title queries. For example, the Indep model improves over SD by 11.27% and 11.7% for Wt10g and Gov2, respectively, more so than that for title queries, with the only exception being the Clue collection description queries. However, the Joint model consistently outperforms both the SD and Indep models in all conditions. These results not only suggest our temporally constrained ranking functions can subsume the SD models, as well as the QL baseline (as illustrated previously), as special tradeoff cases between effectiveness and efficiency, but our models can in fact return more effective results than those commonly used retrieval models under the same time cost.

Chapter 5

A Cascade Ranking Model for Efficient Ranked Retrieval

As we saw in the previous chapters, 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 [2]. 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.

We have discussed how to build efficient ranking models according to a desired tradeoff between efficiency and effectiveness, and how to satisfy the temporal constraint imposed on each query. The central theme of both problems is to control the complexity of the ranking function by using a subset of query-dependent features in the ranking function, thereby reducing query execution time. While efficiency-minded feature selection is a natural way to make existing models faster, such pruning may negatively affect retrieval effectiveness. For example, if there

are hundreds to thousands available query-dependent features, as we impose time constraints on the ranking function, a large fraction of them may need to be eliminated from the ranking function in order to satisfy the speed requirement. 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. Furthermore, 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, 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 this chapter, we ask the question – "is it possible to have *both* speed and high effectiveness at the same time in a ranking model"? We consider the problem of top-K retrieval over web-scale document collections, in which we are interested in returning the best ranked list of K documents to the user such that top-K ranked effectiveness and retrieval efficiency are jointly optimized. There are two observations for this problem: 1) in web-scale collections, the distribution of relevant and non-relevant documents to a query is highly skewed, such that the number of non-relevant documents is usually much larger than the number of relevant documents; 2) the size of the top-K ranked list is usually significantly smaller than the collection size. For instance, in a practical setting, users will only browse through the top-10 or

top-20 returned documents. The naive approach – rank the entire document collection[1] first and then return the top-K, as practiced by conventional ranking models, clearly is quite inefficient since most of the documents evaluated and ranked are outside of the top-K documents. Thus, rather than applying a complex monolithic ranking function, we can perform a quick filtering to remove the most unlikely documents by using simple and less costly term-based ranking functions, and apply more complex and effective ranking models on the most likely documents for the query. This motivates a cascade ranking model for top-K retrieval over web-scale document collections.

We present our proposed cascade models and formally state the problem of learning a cascade to jointly optimize top-K ranked effectiveness and retrieval efficiency. Unlike monolithic ranking models, 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

[1] To be more precise, it is the documents in the postings lists of the query terms.

query execution times. To achieve a desired efficiency-effectiveness tradeoff, we describe a novel boosting algorithm, a generalization of AdaRank [31], 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.

We note that commercial search engines employ a similar multi-stage ranking, which, at the first stage, a basic ranker is applied to get the seed documents, and at the subsequent stages, ranking models of increasing complexity are applied to re-rank and filter these documents. This can be viewed as a baseline case for our cascade model. While this basic architecture offers a way for obtaining speed by progressively filtering documents, there are many unanswered questions/issues:

- To the best of our knowledge, the basic ranker and the more sophisticated ranking models are independently selected in an ad hoc fashion. From an optimization point of view, given a ranked effectiveness/efficiency tradeoff metric on the cascade, such disjoint selection and learning of first-stage and subsequent stage rankers may not lead to an optimal cascade system with respect to the tradeoff metric. It is more desirable to learn the entire cascade as a whole to directly optimize the tradeoff metric.

- Pruning is applied on the ranked documents from the basic ranker to prune documents for the subsequent stage. Clearly, the pruning threshold is critical to both effectiveness and efficiency of the overall system. Setting the thresh-

old too low or too high will result in significantly different tradeoff scenarios between ranked effectiveness/efficiency. What is the criteria for selecting the threshold? What should it optimize?

- Users are only interested in the top-K documents to a query. How will this affect the choice of ranking models and pruning thresholds in the cascade?

Rather than treating multi-stage ranking as an ad hoc application of several ranking models constructed disjointly and with manually selected pruning thresholds, the proposed cascade is a more principled and integrated approach for top-K retrieval over web-scale document collections. We propose to automatically *learn* the ranking models used in the cascade and the pruning thresholds to directly optimize an end-to-end efficiency/effectiveness tradeoff function for the entire cascade. The output of the cascade is a top-K ranked results for each given query, where K is assumed to be given to the system at the training stage.

In recent years, there has been a great deal of research devoted to improving run-time efficiency of machine learned models in various domains, such as object detection and segmentation [73], natural language parsing and translation [74, 75], classification [76] and its applications [77, 78]. In general, there is a fundamental tradeoff between run-time efficiency and effectiveness as models of increasing complexity are considered. Domain-specific cascaded models have been built to alleviate run-time computational costs while trying to ensure result accuracy in these different disciplines. It is worth noting that unlike approximate inference algorithms in machine learning [79, 80], cascaded methods typically involve exact inference in a

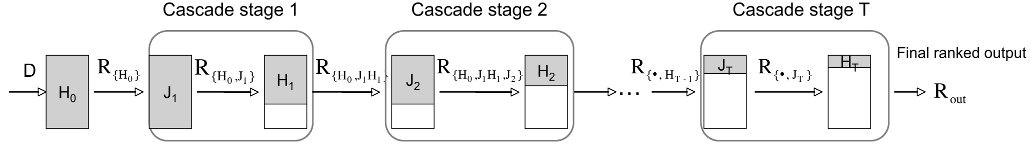


Figure 5.1: An example cascade. After an initial ranking function H_0, each stage consists of two sequential operations: J_t prunes the input ranked documents, then a local ranking function H_t refines the rank order of the retained documents. The new ranked list is passed to the next stage. The size of the shaded area denotes the size of the candidate documents. Subscripts for each ranked list denotes the sequence of actions applied.

reduced input space, rather than approximate inference in full input space. Given this previous work, it might be useful to answer the question – "why do we need a cascade model for efficient ranked retrieval? Can we directly use the existing cascade models and their learning methods for information retrieval?" Although our approach belongs to the general class of coarse-to-fine grained models, it is often desirable to design domain-specific models – cascades used in other disciplines may not have the optimal structure for ranked retrieval and the desired evaluation metric for information retrieval (e.g., MAP, NDCG, etc). For example, cascades used in classification [76, 77] often involve progressive filtering of inputs to reduce classification error and improve run-time efficiency. In contrast, the pruning and re-ranking employed by our cascade for each stage operation allows the model to more effectively improve *ranked* effectiveness measure *and* run-time efficiency.

5.1 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., [81, 49, 8]), 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. Thus, each successive stage is presented with a smaller candidate set, which enables the cascade to exploit more complex and costly ranking models—hence improving effectiveness—without sacrificing efficiency.

As an example, Figure 5.1 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 5.1.1). 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:

$$f_T(d_i) = \sum_{t=1}^{T} \alpha_t \cdot H_t \tag{5.1}$$

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 5.2.

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:

$$|R_{\{\cdot,J_t\}}| \leq |R_{\{\cdot,J_{t-1}\}}| \leq \ldots |R_{\{\cdot,J_1\}}| \tag{5.2}$$

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:

$$E(R_{\{\cdot,H_t\}}) \geq E(R_{\{\cdot,H_{t-1}\}}) \geq \ldots E(R_{\{H_0\}}) \tag{5.3}$$

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 (5.2) or Equation (5.3) 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 5.2.

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.

5.1.1 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 [6] and score-based pruning methods [6, 82] 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

Pruning	Description
Rank-based	$\textsc{RankCutoff}(\beta_t) : (1-\beta_t) \cdot \lvert R_{\{\cdot,H_{t-1}\}} \rvert$
Score-based	$\textsc{ScoreCutoff}(\beta_t) : \beta_t \cdot \textsc{ScoreRange}_{t-1} + \textsc{MinS}_{t-1}$
Score-distribution based	$\textsc{MeanMax}(\beta_t) : \beta_t \cdot \textsc{MaxS}_{t-1} + (1-\beta_t) \cdot \textsc{MeanS}_{t-1}$

Table 5.1: Pruning functions used by our constrained cascade models. Here β_t denotes the pruning threshold, $\mathrm{ScoreRange}_{t-1}$ is defined to be the difference between the maximum and minimum scores in input $R_{\{\cdot,H_{t-1}\}}$, MinS_{t-1} and MaxS_{t-1} are the minimum and mean scores in $R_{\{\cdot,H_{t-1}\}}$, respectively.

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) as shown in Table 5.1 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:

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

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, a 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 [83, 82] 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 [55]. This formulation has the advantage that the pruning function can be better suited for each individual ranked list of documents.

5.2 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 5.1.1), 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.

5.2.1 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:

$$C(\mathbf{S}, q_i) = \sum_{t=1}^{T} C(S_t, q_i) \tag{5.4}$$

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:

$$C(S_t, q_i) = U_t \cdot |R_{i\{\cdot, J_t\}}| \tag{5.5}$$

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 [48, 49]. The most common approach is directly fitting U_t to the actual query execution time of the ranking model [48]. We use this 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 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.

5.2.2 Tradeoff metric

The cascade learning problem is a multi-objective optimization problem [84]. 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 5.4), and $\gamma \in [0, 1]$ is a parameter that controls the relative importance between effectiveness and efficiency. Note $E(\mathbf{S}, q_i)$ 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 5.1); 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, the total cost will increase. Therefore, in order to improve the tradeoff metric, the effectiveness gain from adding extra stages must counteract their costs.

5.2.3 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 5.2.4), pruning functions (Section 5.1.1), 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 [31], that jointly learns the cascade structure and parameters. AdaRank scores documents via a linear sum of ranking models, where each model employs a single feature. 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 [31, 2]. 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 3. The algorithm

Algorithm 3: 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 $\mathbf{S}$

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 5.2.4.

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, $|R_{\{H_0\}}| = 20,000$).

5.2.4 Cascade Stage Construction

In this paper, we use single features as weak rankers H_t, as in AdaRank [31]. Table 3.1 in Chapter 3 provides a summary of the features, which are similar to those in previous work (e.g., [67]). 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 $N \in \{1, 2, 4\}$), and a bigram feature score for unordered terms (parameterized with a window $N' \in \{2, 4, 8\}$).

Typically, bigram features are computed over the entire query, which is problematic, as pointed out in Wang et al. [85, 86]. 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 [34]. 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 (Section 5.1.1) 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:

$$\begin{aligned} S_t = max_{S_t}\ \varphi_t^2 - \left[\sum_{q_i} \frac{P_t(q_i)}{1 - \gamma \cdot C(S_t, q_i)} \right]^2 \\ \text{where } \varphi_t = \sum_{q_i} \frac{P_t(q_i)}{1 - \gamma \cdot C(S_t, q_i)} E(S_t, q_i) \end{aligned} \quad (5.6)$$

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).[2] 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 this work, we employ grid search [72] to find the set of H_t, J_t and β_t that maximizes the equation.

A formal analysis of the boosting algorithm and the proof are presented in Section 5.2.5. 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.

5.2.5 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:

$$\max_{\mathbf{S}} \sum_{q_i} \mathrm{T}(\mathbf{S}, q_i) \tag{5.7}$$

which is equivalent to:

$$\min_{\mathbf{S}} \sum_{q_i} (1 - \mathrm{T}(\mathbf{S}, q_i)) \tag{5.8}$$

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

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

$$\min_{\mathbf{S}} \sum_{q_i} exp(-\mathrm{T}(\mathbf{S}, q_i)) \tag{5.9}$$

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 5.9 is the same as:

$$\min_{S_t} \sum_{q_i} exp(-\mathrm{T}(\mathbf{S}_{t-1} \cup S_t, q_i)) \tag{5.10}$$

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" [87], 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:

$$T \geq 1 - \prod_{t=1}^{T} e^{-\delta^t_{min}} \sqrt{\left[\sum_{q_i} \frac{P_t(q_i)}{1 - \gamma \cdot C(S_t, q_i)}\right]^2 - \varphi_t^2} \tag{5.11}$$

where φ_t is given in Equation 5.6, and let

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

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

$$e^{-\delta^t_{min}} \sqrt{\left[\sum_{q_i} \frac{P_t(q_i)}{1 - \gamma \cdot C(S_t, q_i)}\right]^2 - \varphi_t^2} < 1 \tag{5.12}$$

That is, this condition is satisfied as long as the gain in effectiveness from additional stages is not outweighed by its cost.

Here we prove Equation 5.11. 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 5.2, we get $e^{\alpha_T} = \sqrt{\frac{2\phi_T}{2(\phi_T - \varphi_T)}} = \sqrt{\frac{\phi_T}{(\phi_T - \varphi_T)}}$.

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

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:

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

$$
\begin{aligned}
&\leq \; N\ e^{-\delta^1_{min}} \sqrt{4\phi_1(\phi_1 - \varphi_1)} \prod_{t=2}^{T} e^{-\delta^t_{min}} \sqrt{4\phi_t(\phi_t - \varphi_t)} \\
&= \; N \prod_{t=1}^{T} e^{-\delta^t_{min}} \sqrt{4\phi_t(\phi_t - \varphi_t)}
\end{aligned}
$$

Substitute in ϕ_t and φ_t:

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

So we have:

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

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.

5.3 Experiments

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

5.3.1 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 3.4 in Chapter 3. 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 [31], which can be seen as a special case of the cascade model (i.e., optimized for effectiveness only with no efficiency considerations); the efficient linear model (Chapter 3), which is 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"); and the basic query-likelihood model (QL). For fairness of comparison, "FeaturePrune" re-implements the approach and training method (greedy line search) described in Chapter 3, using the same feature set (Table 3.1) and objective function (i.e., a weighted linear sum of speed and effectiveness) as our cascade model. As previously noted, since Cambazoglu et al. [6] 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.

5.3.2 Implementation Details

All models were implemented in the Ivory open-source retrieval toolkit [64]. 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 [49]. 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 ef-

ficiency 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. In all cases we used a single monolithic inverted index (i.e., no document partitioning). Based on the method described in Section 5.2.1, 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.

5.3.3 Effectiveness vs. Efficiency

Table 5.2 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

	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	33.50	3.90	47.37*	53.60	6.55	**30.94***	**37.40**
FeaturePrune	0.201	34.86	33.10	3.61	47.16	51.87	5.70	29.66	36.20
Cascade	0.175	**35.60**	**33.80**	2.00	**47.44***	**54.47***	4.28	30.60*	**37.40**

Table 5.2: 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.

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 follows from Chapter 3, discovering a better tradeoff point by giving up a bit of effectiveness for a gain in efficiency, except it is optimized with respect to the tradeoff metric defined in this chapter (i.e., a weighted linear sum of speed and effectiveness). However, the cascade model can 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 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

	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 5.3: For each stage of the cascade models in Table 5.2, 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.

also observed for P20.

5.3.4 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 5.3: 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 5.3 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 5.2 (which optimizes the same tradeoff), the cascade quickly achieves comparable effectiveness, and then surpasses it in subsequent stages. For instance, in comparison to Table 5.2, 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 mod-

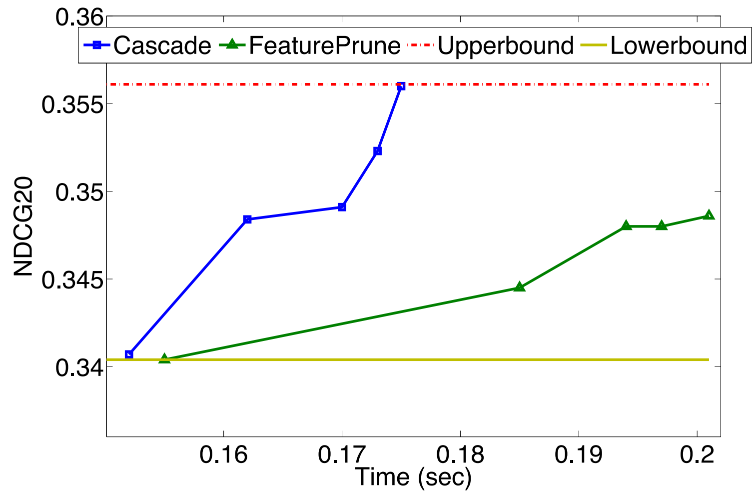
els. This is confirmed by the very low filter loss reported in the table (nearly zero for all stages).

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.

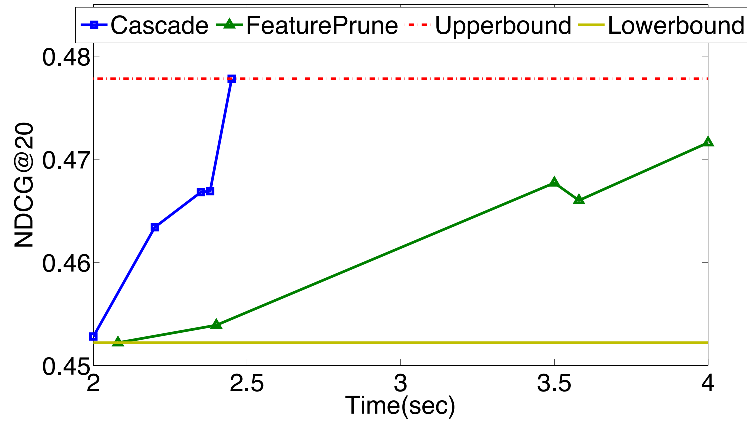
Finally, further analysis reveals that relevant documents that are filtered by our cascade are not ranked in the top k documents by AdaRank either, i.e., these documents have no chance of entering the top k even if an effectiveness-centric model is used. The cascade model is able to obtain the best of 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.

5.3.5 Parameter Variations

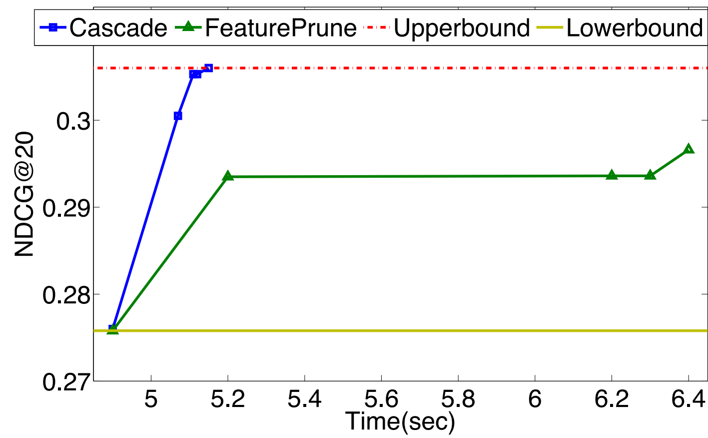
Our final set of experiments explores the effects of varying γ, the tradeoff parameter that balances effectiveness E and cost C in our objective function T.



(a) Dataset: Wt10g



(b) Dataset: Gov2



(c) Dataset: Clue

Figure 5.2: NDCG20 as a function of time, by varying $\gamma \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$.

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 5.2 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).

Chapter 6

Constrained Ranking via Cascades

We addressed the problem of ranking under time constraints in Chapter 4. Our proposed approach relies on reducing the number of ranking features evaluated in order to satisfy a given temporal constraint. Although this approach has the ability to meet time constraints, it focuses exclusively on feature selection and ignores the impact of candidate document set size on efficiency (i.e., how many documents will be scored by the ranking model).

In this chapter, we synthesize our ranking under temporal constraints work with the cascade-based ranking functions. The basic idea behind the cascade model is that many non-relevant documents are needlessly scored during retrieval. A cascade of ranking models, constructed from low to high complexity, is used to progressively filter and refine candidate documents to improve overall efficiency. However, the cascade model in Chapter 5 is unconstrained, which means it cannot return ranked results within a pre-specified amount of time. Therefore, we propose a constrained cascade model for ranking under temporal constraints. Given a time budget, a constrained version of the cascade model is automatically constructed to return the best possible ranked list within the specified time limit. The constrained cascade we propose in this chapter aims to simultaneously optimize both the features used within the model and the document refinement strategy. Thus, the constrained

cascade can better utilize the available budget, via the stage-wise document pruning mechanism of the cascade model, yielding a more robust, more efficient, and more effective retrieval model than the original constrained ranking model [85].

6.1 A Constrained Cascade Model

Given the top-k syndrome of Web search (i.e., users are typically most sensitive to the quality of the top k results and how long it took for the system to retrieve them), and the fact that the number of relevant documents relevant to any given query is typically very small, monolithic models are not efficient for large-scale collections, since a large portion of the candidate documents are non-relevant and/or will not appear in the top-k. Therefore, performing expensive computations over such documents is costly, yet does not ultimately yield substantial relevance gains.

As a quick review, the cascade model in the *unconstrained* case [88] aims to quickly/cheaply filter out most of the non-relevant documents by using inexpensive ranking features, then applying complex/expensive features later onto a smaller set of documents (which are likely to contain the most relevant documents). It is formed by a sequence of additive ranking stages, where each stage consists of a pruning function and a local ranker. The input to each stage is a set of ranked documents that has been filtered and re-ranked by the previous stages. Given stage input, each stage first uses the local pruning function to remove documents in input that have a low chance of appearing in the top k results or being relevant. The resulting smaller set of documents is given to local ranker for re-ranking. The output of each stage is

then passed to the next stage for further processing. Thus, we have a successively reduced set of documents at each stage whose top k ranked effectiveness is iteratively refined.

There are two main challenges in constructing constrained cascades that can return the best possible ranked results (i.e., best top k) within the pre-specified time. First and foremost, how should a constrained cascade be defined? We need to enforce a time requirement and make sure the cascade can indeed satisfy it. Secondly, learning the constrained cascade model is an important task, since most previous learning to rank algorithms focus on optimizing a single metric (i.e., retrieval effectiveness) and cannot be trivially extended to account for a time requirement. Our proposed learning algorithm, which will be described in Section 6.1.2, jointly optimizes the selection of local rankers and pruning functions to maximize effectiveness without violating the time constraint imposed on the model's query execution time. We begin this section by describing the components of the constrained cascade.

6.1.1 Constrained cascade

In general, cascade models can encode different efficiency and effectiveness characteristics. By increasing the number of cascade ranking stages with additional high quality/expensive ranking models and reducing the amount of document pruning, we obtain effectiveness-centric cascade models. By truncating the number of stages and increasing document pruning aggressiveness, we obtain highly efficient cascade models (which is particularly important for web search, as one may be will-

ing to sacrifice effectiveness for improved speed). Between these two extremes there exists a wide range of cascades that exhibit different efficiency/effectiveness tradeoff characteristics. Hence, the ability to impose a time constraint provides a simple way for trading off speed and accuracy.

Given a query q with time requirement T, we say a cascade $W = \{W_1, \ldots, W_N\}$ is a valid constrained cascade if its total query execution time does not exceed T. More formally, it has the following form:

$$S_N(d_i) = \prod_{n=1}^{N} F_n(d_i, q) \quad \sum_{n=1}^{N} \lambda_n \cdot M_n(d_i, q) \tag{6.1}$$

$$s.t. \sum_{n=1}^{N} Cost(W_n(q)) \leq \mathrm{T}$$

Where the first equation denotes the final score assigned to a non-pruned document d_i at the end of the cascade. As mentioned earlier, in this chapter we use the weighted linear combination of local model scores as the final score of each document, where λ_n denotes the local model importance, and $M_n(d_i, q)$ denotes the score assigned to d_i by local ranker M_n. $F_n(d_i, q)$ is a binary value denoting the outcome of the pruning function at stage n for d_i. $F_n(d_i, q)$ is set to one if d_i passes F_n, otherwise, it is zero. Thus, a document reaches the final stage without being pruned if the product of all pruning functions $F_n(d_i, q)$ is equal to one.

In addition, the temporal constraint T on the cascade must be satisfied by making sure the the total execution cost of the cascade W, computed as the sum of execution costs of each stage W_n, does not exceed T. Let $Cost(W_n(q))$ denote the cost of a given stage W_n from evaluating the local model M_n on the reduced set of

documents returned from pruning function F_n. $Cost(W_n(q))$ is determined by the complexity of M_n and how many documents will be scored by M_n. In this work, we employ an analytical model for estimating the actual query execution cost at each stage W_n. Let H_n denote the unit cost of evaluating M_n over each document. The cost at stage W_n is given by:

$$Cost(W_n(q)) = H_n \cdot |R_{nf}(q)| \tag{6.2}$$

where $|R_{nf}(q)|$ denotes the size of the non-pruned documents after applying F_n at stage W_n. Thus, this cost model captures the fact that evaluating a complex model M_n (i.e., large H_n) over a large number of documents will result in higher cost. The value of unit retrieval cost H_n can be determined empirically [48, 49]. In this work, we directly fit H_n to match the actual query execution time of the model M_n over a set of training queries, by setting H_n to be the average time taken by the local ranker M_n for evaluating each document that reaches stage W_n. In our experiments, we will empirically evaluate how well the constrained cascade built with the analytical cost model can *actually* satisfy T.

Following our previous work, we use single ranking features as local rankers M_n [88]. While other local rankers, such as decision trees [51], can also be used, they are more computationally expensive. In contrast, individual ranking features are often easier to compute, and can be highly effective [31]. Table 3.1 in Chapter 3 presents the set of ranking features considered in this work. The feature set consists of both term-based [12] and term-proximity features [15, 21]. Each term-based feature is computed with a scoring function (either BM25 or Dirichlet). The term-

proximity features are defined over the bigrams in each query, and are computed with a BM25 or Dirichlet scoring function, for a specific window type (ordered/unordered) and window length. It should be clear that the term-proximity features are more expensive to compute (due to matching multiple query terms in each document) than the term-based features, although they can be highly effective [15, 21, 18]. Several learning to rank tasks [31, 67] have also used similar types of features. We note the constrained cascade can certainly incorporate other ranking features in addition to those mentioned here – however, since our focus is on the constrained model rather than the feature aspect, exploring the complete space of possible features is beyond the scope of this chapter.

Recall that λ_n is the weight of each local ranker M_n, which denotes how much contribution M_n has on the final score of each document. Intuitively, λ_n should capture the effectiveness of the local model M_n, and it is important to compute an *accurate* λ_n for each local model, since it has a direct impact on the overall effectiveness of the model. In this chapter, we compute λ_n via the parameterization method in Chapters 3 and 4, which assume each λ_n takes on a parametric form with respect to a set of meta-features g_j (Table 3.2).

Finally, Table 5.1 in the previous chapter summarizes the set of pruning functions considered here. As a quick review, we employ this set of pruning functions since they have been shown to be highly effective and have a very low chance of pruning relevant documents that may appear in top k [88, 6]. Each pruning function is parameterized by a pruning threshold β_n. In the rank-based pruning, a proportion of the input documents at each stage is pruned based on their position within the

ranked list. A document is pruned if its ranked position is below the cutoff value. Similarly, in the score-based pruning, a document is pruned if its score is below the score cutoff value, where the cutoff is computed based on the range of document scores in the input set. Finally, the score-distribution based pruning considers the score distribution of the input documents, captured by the mean and max of the input scores. The cutoff is defined as a combination of the mean and the max of the input document scores, and a document is pruned if it scores less than the cutoff. This formulation has the advantage that the pruning function can be better suited for each individual ranked list of documents. As noted before, all pruning functions depend on pruning threshold β_n (between 0 and 1). Large values of β_n lead to more aggressive pruning.

6.1.2 Learning constrained cascade models

In this section, we describe a new algorithm for learning constrained cascades, such that the learned models return the best possible results within time T. Our proposed algorithm is different from our previous work on ranking under temporal constraints, which ignored the impact of the number of documents scored on efficiency and was primarily focused on reducing the number of features evaluated [85]. Our algorithm for learning constrained cascades simultaneously optimizes the selection of documents and ranking features at each stage of the cascade, which can result in more efficient *and* effective models for constrained query execution.

The entire cascade model is represented by $\{F_n(\beta_n), M_n, w_j\}$, where F_n is the

pruning function parameterized by threshold β_n at stage n, M_n is the local ranker, and w_j are the meta-parameters used by the cascade for computing importance weights λ_n. In the learning algorithm, we use NDCG at k as the effectiveness measure, although other metrics defined over top k rankings can be substituted. For retrieval efficiency, our main measure is query execution time (i.e., the amount of time taken to return ranked results). Search engine implementation details (e.g., query evaluation techniques, caching, etc.) are orthogonal to our task, since their effects on query execution time can be captured by our cost model for query execution time, which simply serves as an input to the learning algorithm.

When learning a constrained cascade, there are two competing measures:

1) Ranked effectiveness (γ): maximize the top k ranked effectiveness at the cascade output.

2) Retrieval efficiency (σ): maximize the number of pruned documents at each stage to improve efficiency and ensure the time constraint can be satisfied.

More formally, the learning objective is stated as a joint optimization over model parameters $\{F_n(\beta_n), M_n, \lambda_j\}$.

$$\max_{F_n(\beta_n), M_n, w_j} \gamma(q) \; s.t. \; \sigma(q) \leq \mathrm{T} \tag{6.3}$$

Where γ and σ denote effectiveness and efficiency, respectively, q denotes a query, and T denotes the time constraint.

Algorithm 4 presents our solution for this learning problem. The problem is solved with a two-step procedure. First, given the set of ranking features M_n, we learn the meta-feature weights w_j (which are used to compute feature weights in

Algorithm 4: Learning a constrained cascade model

Input: Query execution time requirement T

Output: Temporally constrained cascade W

Initialize constrained cascade $W = \{\}$;

Initialize $totalCost = 0$;

Learn meta-feature weights w_j;

while $totalCost \leq$ T **do**

  for $W_n =< M_n, F_n, \beta_n >$ **do**

    $Score(W_n) = \gamma(W \cup W_n) - \gamma(W)$;

    Compute profit density $p(W_n) = \frac{Score(W_n)}{Cost(W_n)}$;

  end

  $W_n^* = \arg\max_{W_n} p(W_n)$;

  $W = W \cup W_n^*$;

  $totalCost = totalCost + Cost(W_n^*)$;

end

return constrained cascade W;

the cascade ranking model) to maximize ranked effectiveness for several settings of time constraints T. Note under each setting of time T, it is only permissible to use a subset of ranking features. So the goal of this step is to identify weights w_j that can be effective for a wide range of time requirements. Standard learning to rank algorithms can be used to learn w_j. We employ a simple, yet highly effective coordinate-ascent algorithm [72]. It iteratively optimizes the averaged effectiveness over the time constraints by performing a series of one-dimensional line searches in

the meta-feature weight space. At each iteration, it searches for an optimal value for w_j while holding all other meta-feature weights fixed. This iterative process continues until the improvement in the objective metric drops below a threshold.

In the second step, given the learned meta-feature weights w_j, we learn the remaining parameters of the constrained cascade (i.e., local model M_n, pruning function and threshold $F_n(\beta_n)$ at each stage). This step takes an iterative selection approach, which proceeds in rounds to sequentially learn a set of stages to optimize ranked effectiveness before running out of time budget. It should be easy to see this step corresponds to the classic knapsack problem. Here, pairs of M_n and $F_n(\beta_n)$ are the "items" that we are trying to fit into the knapsack. The value of each item is determined by how much improvement can be obtained for effectiveness if the pair is added to the constrained cascade (which is given as the score of the pair), and the cost of the pair M_n and $F_n(\beta_n)$ is simply the evaluation cost of applying M_n over R_{nf} (i.e., the documents that passed pruning function $F_n(\beta_n)$), as given by Equation 6.2 (Section 6.1.1), and the knapsack capacity is T. Intuitively, we want to jointly select the best ranking feature *and* the documents for evaluation to effectively utilize the available time budget.

More specifically, we first compute profit density for each item W_n (a combination of M_n, F_n, and β_n), which is defined by the ratio between its score and cost. The profit density captures two competing factors: the effectiveness gain and the cost from using a candidate stage. The candidate stage with the best profit density is then added to the model. Intuitively, ranking models M_n that are effective over a small set of documents (as determined by $F_n(\beta_n)$) are more desirable in the con-

	Wt10g	**Gov2**	**Clue**
Query exec. time	0.17s	2.09s	4.10s

Table 6.1: Query execution time of baseline QL for title queries of Wt10g, Gov2, and Clue.

strained setting (larger profit density value). The algorithm continues adding stages according to this density (largest first), until it can no longer add any more features without exceeding the time constraint.

Our algorithm can be viewed as a generalization of our previously proposed temporally constrained ranking model [85], since the model proposed here is equivalent to that model when no document pruning is used. As we discussed earlier, document pruning is a key benefit of our proposed model, since it can be both highly effective and efficient since the cascade will not waste time on a potentially large number of non-relevant documents.

We note that our algorithm only requires a linear (in the size of $|M_n|$) number of single parameter training steps each iteration (the number of choices for β_n and F_n are fixed). If we construct N stages before exhausting T, such that $N << |M_n|$, then it is likely that our algorithm will be much more computationally efficient than training a monolithic model with $|M_n|$ features.

6.2 Experiments

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

6.2.1 Experimental setup

We report our results using a diverse set of document collections. We use three TREC web collections as shown in Table 3.4 in Chapter 3. Wt10g is a small web collection with 1.7 million documents, Gov2 is a larger 25 million page crawl of the .gov domain, and Clue is the first English segment of ClueWeb09, a recently-released web crawl consisting of 50 million documents. The title portions of the TREC topics are used as queries and are divided into a training and test set of equal size. All parameter tuning was performed on the training set, and all of the results reported are from applying the learned parameters to the test set.

We compare our proposed constrained cascade model against two other models: one is the standard query likelihood model [12] (QL), with Dirichlet smoothing parameter $\mu = 1000$; and the temporally constrained model in Chapter 4. Two algorithms were proposed in Chapter 4 – the "Independent" and "Joint" feature selection models. The "Joint" model accounts for feature correlations and redundancies, which yielded substantially better results. Thus, to form a competitive baseline, we use the "Joint" model to compare against our proposed approach. For fairness of comparison, the "Joint" feature-based model uses the same feature set

and objective metric as the constrained cascade model. In addition, following our previous work, the time requirement is defined as a multiple m of the average query execution time of the baseline QL over training queries, denoted by $m \cdot T_{ql}$, where the baseline QL query execution times are shown in Table 6.1 for reference [85]. We note the reported retrieval efficiency of our constrained cascade model accounts for the *overall* time taken by the cascade to return query results (e.g., accounts for time taken by all stages for local model evaluation and pruning operation).

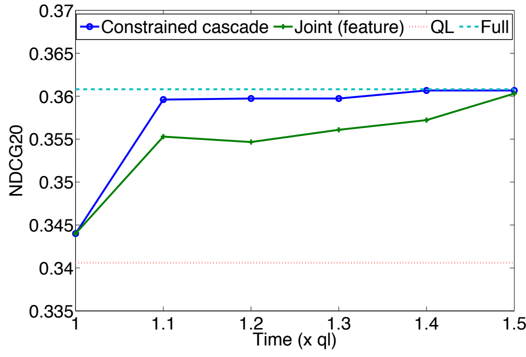
We also construct an effectiveness-centric model ("Full" model) which is optimized for effectiveness only, and use it as the effectiveness upper-bound for the constrained models. It is constructed in the same manner as the constrained cascade models, except the time constraint is set to "infinity" for learning this model and document pruning is not performed. We use NDCG20 and precision at rank 20 (P20) as the primary effectiveness measures, although a variety of other effectiveness metrics for top k can be substituted.

6.2.2 Results

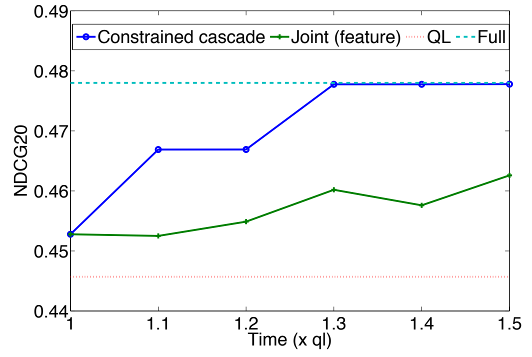
We are now ready to describe the outcome of our experimental evaluation.

6.2.2.1 Ranked effectiveness vs time constraints

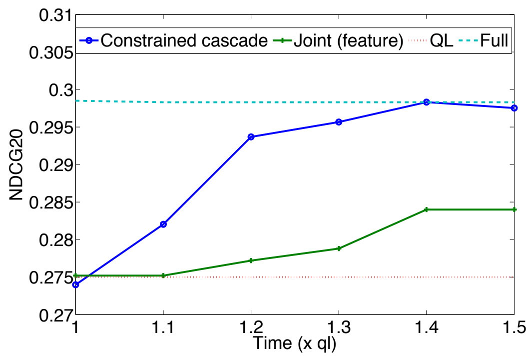
Figure 6.1 compares the NDCG20 and P20 of our proposed constrained cascade and the Joint model for temporally constrained ranking as a function of time constraints, from $1 \cdot T_{ql}$ to $1.5 \cdot T_{ql}$ in increments of $0.1 \cdot T_{ql}$. The effectiveness of the



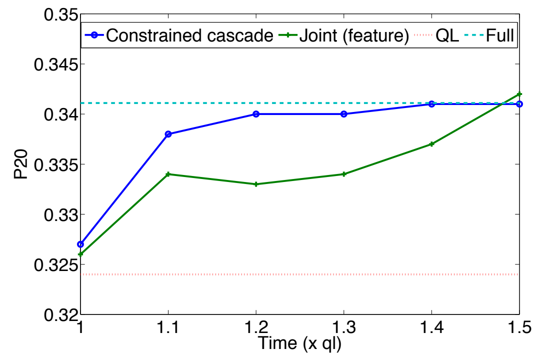
(i) Wt10g NDCG20



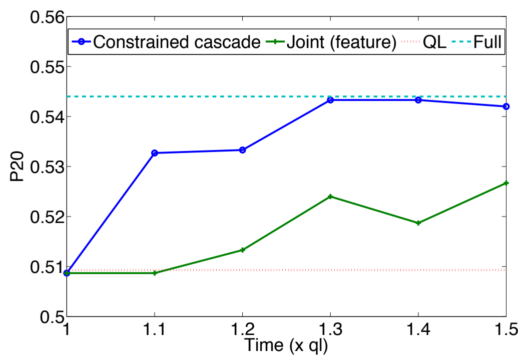
(ii) Gov2 NDCG20



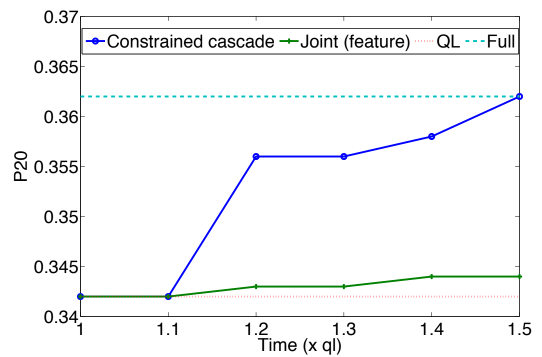
(iii) Clue NDCG20



(iv) Wt10g P20



(v) Gov2 P20



(vi) Clue P20

Figure 6.1: NDCG20 and P20 versus time requirements for the Constrained cascade and "Joint" (feature-based) models for test queries in Wt10g, Gov2 and Clue.

baseline QL is plotted as the lower dotted line, and the "Full" model effectiveness upper-bound is plotted as the upper dotted line. What is not shown in the figure is that the time taken by the effectiveness-centric upper-bound ("Full" model) is roughly $5 \cdot T_{ql}$ for each dataset. Thus, although the upper-bound is very effective, it is highly inefficient.

Although both of the constrained cascade and Joint (feature-based) models can produce more effective results than baseline QL when the time constraint is relaxed, the constrained cascade model consistently outperforms (for both NDCG20 and P20) than the Joint model across a wide range of time constraints for all datasets. As compared to the final effectiveness achieved by the Joint model, the constrained cascade can quickly obtain the same effectiveness, and then surpass it at subsequent time points. For instance, in five out of six cases, by time constraint $1.1 \cdot T_{ql}$, the constraint cascade surpasses the final score of the feature-based method (Joint), while the feature-based method takes a considerately longer amount of time to achieve the same effectiveness (it is nearly 30% slower than the constrained cascade). This illustrates that the constrained cascade is highly effective compared to the Joint model when strict time constraints are imposed. This also shows the advantage of the constrained cascade – by progressively reducing the number of candidate documents, it facilitates the use of high quality ranking functions for effectiveness gains while being able to simultaneously satisfy tight computational constraints.

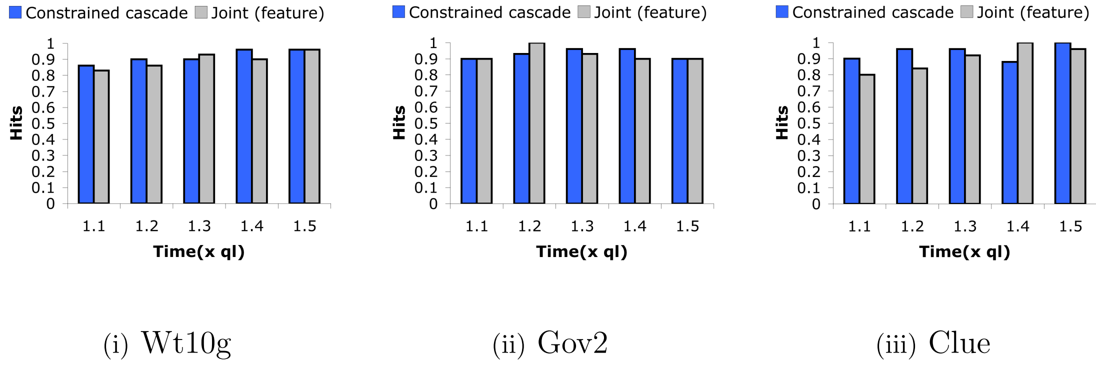
We also observe the effectiveness difference between the constrained cascade and Joint is largest for Gov2 and Clue (i.e., the two largest collections). This highlights that jointly selecting documents and local ranking features for temporally

constrained ranking, as considered by the constrained cascade model, is particularly useful for large-scale document collections (in which a higher percentage of documents may be non-relevant/noisy). Focusing on a small set of the most likely relevant documents not only reduces computational cost, but can also lead to more effective results by eliminating many noisy documents from consideration. This also demonstrates monolithic ranking functions (i.e., the feature-based model) may waste too much computation on documents that are unlikely to appear in the top k. More critically, such wasteful computation can quickly exhaust the time budget and thus limits the application of additional ranking features for effectiveness gain. This last point is further illustrated by the feature-based method's very slow rate of improvement for effectiveness even as the time constraint is significantly relaxed.

Finally, as compared to the effectiveness upper-bound ("Full" model), the constrained cascade approaches the upper-bound very rapidly as more time is given. By time $1.4 \cdot T_{ql}$, the constrained cascade achieves the NDCG of the "Full" model. Similar results are also observed for P20. This shows that the constrained cascade can equal an effectiveness-centric model, while simultaneously being able to satisfy tight time constraints.

6.2.2.2 Ensuring time requirements

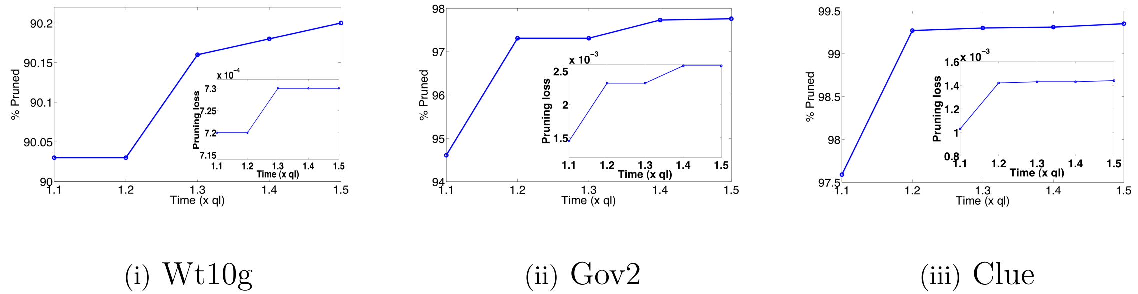
Our learning algorithm uses an analytical cost model to estimate the actual query execution time of the constrained cascade to ensure it does not exceed the time constraint. In this section, we investigate whether time constraints can actually



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

Figure 6.2: Bar chart showing the fraction of query evaluation times that satisfy the imposed time constraint for Wt10g, Gov2, and Clue.

be met in practice. Figure 6.2 illustrates how well the constrained cascade meets the time constraints for each dataset. Results for the feature-based model (Joint) are also reported. Each bar represents a time point, and the height of the bar (i.e., hits) denotes the fraction of queries that actually satisfy the time constraint (i.e., the time constraint is satisfied if the constrained model returns results within the targeted time budget). This set of figures shows that both of the constrained cascade and the feature-based models can satisfy the time constraints quite well — above 90% for almost all collections. This suggests our analytical cost model, as used by the algorithm for estimating the actual query execution time of the cascade, works reasonably well for ensuring time constraints are satisfied in practice. We hypothesize that a slightly more complex analytical cost model would easily be able to satisfy the time constraints even better.



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

Figure 6.3: % documents pruned as a function of time constraints, and the corresponding pruning loss (insert).

6.2.2.3 Document pruning and pruning loss

In this section, we examine the document pruning operation performed by the constrained cascade and the resulting pruning loss – defined to be the percentage of relevant documents incorrectly pruned out of all candidate documents considered by the constrained cascade (where the candidate documents are produced by the initial ranker R_0). Recall the constrained cascade satisfies the time constraint by jointly optimizing the selection of ranking feature and documents at each cascade stage. Since the constrained cascade is flexible enough to use potentially different pruning functions at different stages (i.e., the selection of the pruning function depends on its combined performance with the local model), our focus is not on studying the details of each pruning function, but rather on the combined performance of the learned pruning functions for the entire model. Figure 6.3 presents the overall percentage of documents pruned by the constrained cascade and the resulting pruning loss as a function of time constraints. From this figure, we see that the number of documents pruned is positively correlated with the time budget since more documents get

pruned as more stages of the model are added. The figure also shows that the amount of pruning is well above 90% for all time constraints, which yields a significant reduction in the computational cost associated with each local model.

While pruning helps to control the overall efficiency of the constrained cascade, it is important to ensure it has very little, if any, impact on the effectiveness of the model. As shown from the pruning loss figures (Figure 6.3 insert), pruning has a very low chance of removing relevant documents from the candidate document set (i.e., it is nearly zero for all time constraints across all datasets). Further analysis reveals that relevant documents that are filtered by our constrained cascade are not ranked in the top k documents by the "Full" (effectiveness-centric) model either, i.e., these documents have no chance of entering the top k even if an effectiveness-centric model is used. Thus, pruning proves to be very useful, yielding better efficiency and having minimal impact on effectiveness, which enables the constrained model to simultaneously maximize top k effectiveness *and* ensure the time requirement.

6.2.2.4 Average effectiveness across time requirements

The previous section demonstrated our constrained cascade outperforms the best previously known constrained model over different time constraints. In this section we investigate the average effectiveness of the constrained models across the time points (from times $1 \cdot T_{ql}$ to $1.5 \cdot T_{ql}$). We compare the average effectiveness of the constrained cascade and the Joint (feature-based) model in Table 6.2. Results from QL are also included.

	Wt10g		Gov2		Clue	
	$NDCG20_{AVG}$	$P20_{AVG}$	$NDCG20_{AVG}$	$P20_{AVG}$	$NDCG20_{AVG}$	$P20_{AVG}$
QL	34.06	32.40	44.57	50.93	27.50	34.20
Joint (feature)	35.46 (+4.1)	33.46 (+3.3)	45.68 (+2.5)	51.66 (+1.4)	27.91 (+1.5)	34.30 (+0.3)
Constr. cascade	**35.74*** (+4.9/0.8)	**33.78** (+4.3/1.0)	**47.00*** (+5.5/2.9)	**53.39**$^{*}_{\dagger}$ (+4.8/3.3)	**29.02** (+5.5/4.0)	**35.27*** (+3.1/2.8)

Table 6.2: Average effectiveness in terms of averaged NDCG20 and averaged P20 from times $1 \cdot T_{ql}$ to $1.5 \cdot T_{ql}$ for Wt10g, Gov2, and Clue. Bolded values denote best performance obtained in each dataset. The $*$ and $\dagger$ symbols represent statistically significant differences with respect to QL and Joint (feature-based) model, respectively. Percentage improvement shown in parentheses: over QL for Joint, and over QL/Feature-based for Constrained cascade.

In terms of average effectiveness computed using $NDCG$, the constrained cascade consistently outperforms both feature-based model and QL. For instance, the constrained cascade has 4.9%, 5.5%, and 5.5% improvement over baseline QL on Wt10g, Gov2, and Clue, respectively. The constrained cascade achieves 0.8%, 2.9% and 4.0% improvement over the Joint model on the same collections, respectively. Similar results are also achieved under averaged P20. While the Joint model (feature-based) can also outperform QL in terms of the averaged effectiveness, we note it is not able to attain the same level of improvements as the constrained cascade, and none of the improvements by the Joint model over QL is statistically significant, whereas the constrained cascade outperforms QL with statistical significance in two out of three collections, using both effectiveness metrics.

We further observe the constrained cascade achieves larger effectiveness gains over QL and the feature-based model for Gov2 and Clue, two of the largest collections, as compared to the smaller Wt10g collection. For instance, in terms of the average effectiveness computed using NDCG, the constrained cascade outperforms QL by 5.5% for both Gov2 and Clue, as compared to 4.9% for Wt10g. Similarly, constrained cascade improves over the feature-based model by 2.9% and 4.0% for Gov2 and Clue, respectively, as compared to 0.8% for Wg10g. This confirms our earlier observation that our constrained cascade is particularly beneficial for large-scale corpora — under tight time budgets, discarding a large amount of non-essential documents (as existed in these large collections) allows the model to fit in additional ranking stages to improve effectiveness without hurting the time constraints.

Chapter 7

Conclusion

We have presented a principled and unified framework for learning ranking functions that jointly optimizes 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 introduced new classes of ranking models (efficient linear models, temporal constrained models, cascade ranking functions, etc), optimization metrics, and learning algorithms to construct fast and effective ranking models. We have shown that they subsume commonly-used ranking models as special tradeoff cases that encode fixed points in the effectiveness/efficiency solution space.

7.1 Limitations

The current limitations of this work include:

1. We have limited the feature pool to simple term-based and term-proximity features, and the space of models to linear models. However, there can be a large variety of different features (e.g., user-based), and different classes of ranking models (e.g., boosted regression trees). While our general framework

can deal with any number of features as inputs, the learning algorithms focus on linear models only.

2. We have only focused on two forms of tradeoff metrics (e.g., harmonic means for efficient linear models, and linear combinations for cascades), and have not shown how the learned models may change as a result of combining speed and effectiveness differently (e.g., beyond harmonic means and linear combinations). Thus, results reported are limited to our particular choices in the tradeoff metric and can not be used to demonstrate the model performance under different tradeoff metric definitions.

3. The line-search algorithm employed by efficient linear models can be costly for large training sets and large feature sets due to its non-analytical nature. As the number of training features increases, the training efficiency can be a serious concern in our current experiment framework. One way to alleviate this issue is that we can use other approaches for directly optimizing non-smooth functions that use fewer function evaluations, such as simultaneous perturbation stochastic approximation [63].

7.2 Future work

The learning to efficiently rank framework can lead to several promising future work directions in devising practical search engine ranking models. They include:

1. It would be very interesting to extend our learning algorithms to work with

boosted regression trees [1], which have been shown to be one of the best learning to rank models.

2. By enriching temporally constrained ranking models with user/query-specific time budgets, we can assign larger time budgets for more commercially viable queries, which will translate into better search results and more ad clicks for these queries.

3. Many other search criteria exist in addition to speed and effectiveness (e.g., freshness, diversity, etc). We can apply the intuitions gained from learning to efficiently rank, and utilize divide-and-conquer techniques to jointly optimize the different metrics in a *unified* framework.

4. The class of probabilistic graphical models, which is used to construct temporally constrained ranking functions in Chapter 4, can be used to explore more complex interactions between features and feature costs (e.g., shared feature computations).

5. We have considered applying line-search and boosting algorithms to optimizing harmonic means and linear combinations of speed and effectiveness, respectively. However, in a general setting, the space of all possible selections of optimization metrics and learning algorithms can be quite large. Thus, it would be very interesting to have an automatic way for selecting the best pair of optimization metric/learning algorithm for learning to rank.

Bibliography

[1] Chris Burges. From ranknet to lambdarank to lambdamart: An overview. In *Microsoft Research Technical Report MSR-TR-2010-82*, 2010.

[2] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3), 2009.

[3] Paul N. Bennett, Filip Radlinski, Ryen W. White, and Emine Yilmaz. Inferring and using location metadata to personalize web search. In *Proc. 34th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 135–144, 2011.

[4] Ryen W. White, Paul N. Bennett, and Susan T. Dumais. Predicting short-term interests using activity-based search context. In *Proc. 19th Intl. Conf. on Information and Knowledge Management*, pages 1009–1018, 2010.

[5] Kevyn Collins-Thompson, Paul N. Bennett, Ryen W. White, Sebastian de la Chica, and David Sontag. Personalizing web search results by reading level. In *Proc. 20th Intl. Conf. on Information and Knowledge Management*, pages 403–412, 2011.

[6] B. Cambazoglu, Hugo Zaragoza, Olivier Chapelle, Jiang Chen, Ciya Liao, Zhaohui Zheng, and Jon Degenhardt. Early exit optimizations for additive machine learned ranking systems. In *Proc. 3rd ACM Conf. on Web Search and Data Mining*, pages 411–420, 2010.

[7] Vo Anh, Owen de Kretser, and Alistair Moffat. Vector-space ranking with effective early termination. In *Proc. 24th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 35–42, 2001.

[8] Trevor Strohman, Howard Turtle, and W. Bruce Croft. Optimization strategies for complex queries. In *Proc. 28th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 219–225, 2005.

[9] Ricardo Baeza-Yates, Aristides Gionis, Flavio Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. The impact of caching on search engines. In *Proc. 30th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 183–190, 2007.

[10] Na Dai, Milad Shokouhi, and Brian D. Davison. Learning to rank for freshness and relevance. In *Proc. 34th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 95–104, 2011.

[11] Krysta M. Svore, Maksims N. Volkovs, and Christopher J.C. Burges. Learning to rank with multiple objective functions. In *Proc. 19th Intl. Conf. on World Wide Web*, pages 367–376, 2011.

[12] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. In *Proc. 21st Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 275–281, 1998.

[13] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proc. 24th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 334–342, 2001.

[14] Tao Tao, Xuanhui Wang, Qiaozhu Mei, and ChengXiang Zhai. Language model information retrieval with document expansion. In *Proc. of the 2006 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 407–414, 2006.

[15] Donald Metzler and W. Bruce Croft. A Markov Random Field model for term dependencies. In *Proc. 28th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 472–479, 2005.

[16] Michael Bendersky and W. Bruce Croft. Discovering key concepts in verbose queries. In *Proc. 31st Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 491–498, 2008.

[17] Jianfeng Gao, Haoliang Qi, Xinsong Xia, and Jian-Yun Nie. Linear discriminant model for information retrieval. In *Proc. 28th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 290–297, 2005.

[18] Tao Tao and ChengXiang Zhai. An exploration of proximity measures in information retrieval. In *Proc. 30th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 295–302, 2007.

[19] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[20] Matthew Lease. An improved Markov Random Field model for supporting verbose queries. In *Proc. 32nd Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 476–483, 2009.

[21] Stefan Büttcher, Charles Clarke, and Brad Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proc. 29th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 621–622, 2006.

[22] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, 40(2):1–60, 2008.

[23] Anni Coden, Eric Brown, and Savitha Srinivasan. *Information Retrieval Techniques for Speech Applications*. Springer, 2002.

[24] Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[25] F. Wilfrid Lancaster and Emily Gallup. *Information Retrieval On-Line.* Melville Publishing Co., 1973.

[26] Amnon Shashua and Anat Levin. Ranking with large margin principle: Two approaches. In *Proc. 15th Proc. of Advances in Neural Information Processing Systems*, pages 937–944, 2002.

[27] Koby Crammer and Yoram Singer. Pranking with ranking. In *Proc. 15th Proc. of Advances in Neural Information Processing Systems*, pages 641–647, 2002.

[28] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proc. 22nd Proc. Intl. Conference on Machine Learning*, pages 89–96, 2005.

[29] Ming feng Tsai, Tie-Yan Liu, Tao Qin, Hsin-Hsi Chen, and Wei-Ying Ma. Frank: a ranking method with fidelity loss. In *Proc. 30th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 383–390, 2007.

[30] Corinna Cortes, Mehryar Mohri, and Ashish Rastogi. Magnitude-preserving ranking algorithms. In *Proc. 24th Proc. Intl. Conference on Machine Learning*, pages 169–176, 2007.

[31] Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proc. 30th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 391–398, 2007.

[32] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proc. 24th Proc. Intl. Conference on Machine Learning*, pages 129–136, 2007.

[33] Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proc. 30th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 271 – 278, 2007.

[34] Michael Bendersky, Donald Metzler, and W. Bruce Croft. Learning concept importance using a weighted dependence model. In *Proc. 3rd ACM Conf. on Web Search and Data Mining*, pages 31–40, 2010.

[35] Jianfeng Gao, Jian-Yun Nie, Guangyuan Wu, and Guihong Cao. Dependence language model for information retrieval. In *Proc. 27th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 170–177, 2004.

[36] R. Nallapati. Discriminative models for information retrieval. In *Proc. 27th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 64–71, 2004.

[37] Chris Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[38] James Allan, Javed Aslam, Ben Carterette, Virgil Pavlu, and Evangelos Kanoulas. Million query track 2008 overview. In *Proc. 17th Text REtrieval Conference*, 2008.

[39] Charles Clarke, F. Scholar, and Ian Soboroff. Overview of the TREC 2005 terabyte track. In *Proc. 14th Text REtrieval Conference*, 2005.

[40] Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proc. 8th Intl. ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 133–142, 2002.

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

[42] David Carmel, Doron Cohen, Ronald Fagin, Eitan Farchi, Michael Herscovici, Yoelle Maarek, and Aya Soffer. Static indexing pruning for information retrieval systems. In *Proc. 24th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 43–50, 2001.

[43] Stefan Buttcher and Charles Clarke. Efficiency vs. effectiveness in terabyte-scale information retrieval. In *Proc. 13th Text REtrieval Conference*, 2005.

[44] Stefan Buttcher, Charles Clarke, and Peter Yeung. Indexing pruning and result reranking: Effects on ad-hoc retrieval and named page finding. In *Proc. 15th Text REtrieval Conference*, 2006.

[45] Vo Ngoc Anh and Alistair Moffat. Pruned query evaluation using pre-computed impacts. In *Proc. 29th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 372–379, 2006.

[46] Alexandros Ntoulas and Junghoo Cho. Pruning policies for two-tiered inverted index with correctness guarantee. In *Proc. 30th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 191–198, 2007.

[47] Michael Bendersky, W. Bruce Croft, and David A. Smith. Two-stage query segmentation for information retrieval. In *Proc. 32nd Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2009.

[48] Fidel Cacheda, Vassilis Plachouras, and Iadh Ounis. A case study of distributed information retrieval architectures to index one terabytes of text. *Information Processing and Management*, 41:1141–1161, 2004.

[49] Howard Turtle and James Flood. Query evaluation: strategies and optimizations. *Information Processing and Management*, 31(6):831–850, 1995.

[50] Alistair Moffat and Justin Zobel. Self-indexing inverted files for fast text retrieval. volume 14, pages 349–379, 1996.

[51] Zhaohui Zheng, Hongyuan Zha, Tong Zhang, Olivier Chapelle, Keke Chen, and Gordon Sun. A general boosting method and its application to learning ranking functions for web search. In *Proc. 22nd Proc. of Advances in Neural Information Processing Systems*, pages 1697–1704, 2008.

[52] Paul Viola and Michael Jones. Robust real-time object detection. *International Journal of Computer Vision*, 57(2):137–154, 2002.

[53] Chris Burges, Robert Ragno, and Quoc Viet Le. Learning to rank with nonsmooth cost functions. In *Proc. 19th Proc. of Advances in Neural Information Processing Systems*, pages 193–200, 2006.

[54] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. Softrank: optimizing non-smooth rank metrics. In *Proc. 1st ACM Conf. on Web Search and Data Mining*, pages 77–86, 2008.

[55] David Weiss and Ben Taskar. Structured prediction cascades. In *Proc. 13th Conference on Artificial Intelligence and Statistics*, pages 916–923, 2010.

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

[57] Kevyn Collins-Thompson and James Callan. Query expansion using random walk models. In *Proc. 14th Intl. Conf. on Information and Knowledge Management*, pages 704–711, 2005.

[58] Lidan Wang, Paul N. Bennett, and Kevyn Collins-Thompson. Robust ranking models via risk-sensitive optimization. In *Proc. 35th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 275–281, 2012.

[59] Jiang Bian, Xin Li, Fan Li, Zhaohui Zheng, and Hongyuan Zha. Ranking specialization for web search: a divide-and-conquer approach by using topical ranksvm. In *Proc. 19th Intl. Conf. on World Wide Web*, pages 131–140, 2010.

[60] Jiang Bian, Tie-Yan Liu, Tao Qin, and Hongyuan Zha. Ranking with query-dependent loss for web search. In *Proc. 3rd ACM Conf. on Web Search and Data Mining*, pages 141–150, 2010.

[61] Xiubo Geng, Tie-Yan Liu, Tao Qin, Andrew Arnold, Hang Li, and Heung-Yeung Shum. Query dependent ranking using k-nearest neighbor. In *Proc. 31st Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 115–122, 2009.

[62] Xiubo Geng, Tie-Yan Liu, Tao Qin, and Hang Li. Feature selection for ranking. In *Proc. 30th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 407–414, 2007.

[63] Yisong Yue and Christopher Burges. On using simultaneous perturbation stochastic approximation for IR measures, and the empirical optimality of lambdarank. In *NIPS Machine Learning for Web Search Workshop*, 2007.

[64] Jimmy Lin, Donald Metzler, Tamer Elsayed, and Lidan Wang. Of ivory and smurfs: Loxodontan mapreduce experiments for web search. In *Proc. 18th Text REtrieval Conference*, 2009.

[65] Thomas Dean and Mark Boddy. Time-dependent planning. In *Proc. 17th Nat. Conf. on Artificial Intelligence*, pages 49–54, 1988.

[66] Shlomo Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.

[67] Donald Metzler. Automatic feature selection in the Markov Random Field model for information retrieval. In *Proc. 16th Intl. Conf. on Information and Knowledge Management*, pages 253–262, 2007.

[68] Michael Jordan. Graphical models. *Statistical Science*, 19(1):140–155, 2004.

[69] Dan Roth and Wen tau Yih. Integer linear programming inference for conditional random fields. In *Proc. 22nd Proc. Intl. Conference on Machine Learning*, pages 736–743, 2005.

[70] David Edwards. *Introduction to Graphical Modeling*. Springer, 2000.

[71] Michael Jordan, Zoubin Ghahramani, Tommi Jaakola, and Lawrence Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[72] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer science + business media, 2006.

[73] Jonghwan Kim, Chung-Hee Lee, Young-Chul Lim, and Soon Kwon. Stereo vision-based improving cascade classifier learning for vehicle detection. In *ISVC Proceedings of the 7th International Conference on Advances in Visual Computing*, pages 387–397, 2011.

[74] David Weiss, Benjamin Sapp, and Ben Taskar. Sidestepping intractable inference with structured ensemble cascades. In *Proc. 24th Proc. of Advances in Neural Information Processing Systems*, 2010.

[75] Slav Petrov, Ari Haghighi, and Dan Klein. Coarse-to-fine syntactic machine translation using language projections. In *NIPS Coarse-to-Fine Learning and Inference Workshop*, 2010.

[76] Vikas Raykar, Balaji Krishnapuram, and Shipeng Wu. Designing efficient cascaded classifiers: tradeoff between accuracy and cost. In *Proc. 16th Intl. ACM SIGKDD Conf. on Knowledge Discovery and Data Mining*, pages 853–860, 2010.

[77] Jay Pujara and Lise Getoor. Coarse-to-fine, cost-sensitive classification of email. In *NIPS Coarse-to-Fine Learning and Inference Workshop*, 2010.

[78] Jay Pujara, Ben London, and Lise Getoor. Reducing label cost by combining feature labels and crowdsourcing. In *ICML Workshop on Combining Learning Strategies to Reduce Label Cost*, 2011.

[79] Ofer Meshi, David Sontag, Tommi Jaakkola, and Amir Globerson. Learning efficiently with approximate inference via dual losses. In *Proceedings of the 27th International Conference on Machine Learning*, pages 783–790, 2010.

[80] Alex Kulesza and Fernando Pereira. Structured learning with approximate inference. In *Proc. 22nd Proc. of Advances in Neural Information Processing Systems*, 2008.

[81] Eric Brown. Fast evaluation of structured queries for information retrieval. In *Proc. 18th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 30–38, 1995.

[82] Avi Arampatzis, Jaap Kamps, and Stephen Robertson. Where to stop reading a ranked list. In *Proc. 32nd Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 524–531, 2009.

[83] Evangelos Kanoulas, Virgil Pavlu, Keshi Dai, and Javed Aslam. Modeling the score distribution of relevant and non-relevant documents. In *Proc. 2nd Conference on Theory of Information Retrieval*, pages 152–163, 2009.

[84] Ralph E. Steuer. *Multiple Criteria Optimization: Theory, Computation, and Application*. John WIley and Sons, Inc., 1986.

[85] Lidan Wang, Donald Metzler, and Jimmy Lin. Ranking under temporal constraints. In *Proc. 19th Intl. Conf. on Information and Knowledge Management*, pages 79–88, 2010.

[86] Lidan Wang, Jimmy Lin, and Donald Metzler. Learning to efficiently rank. In *Proc. 34th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 138–145, 2010.

[87] Trevor Hastie, Roert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer, 2009.

[88] Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *Proc. 34th Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, pages 275–281, 2011.