9-2011

Query-Dependent Selection of Retrieval Alternatives

Niranjan Balasubramanian
University of Massachusetts - Amherst, niranjan@cs.umass.edu

QUERY-DEPENDENT SELECTION OF RETRIEVAL ALTERNATIVES

A Dissertation Presented

by

NIRANJAN BALASUBRAMANIAN

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2011

Department of Computer Science

QUERY-DEPENDENT SELECTION OF RETRIEVAL ALTERNATIVES

A Dissertation Presented

by

NIRANJAN BALASUBRAMANIAN

Approved as to style and content by:

James Allan, Chair

W. Bruce Croft, Member

David Jensen, Member

Michael Lavine, Member

Andrew G. Barto, Department Chair
Department of Computer Science

ACKNOWLEDGMENTS

I owe the most to my wife, Aruna. She has been a colleague, a friend, and a pillar of support throughout my graduate studies. I am thankful for the opportunity to have pursued our graduate studies together. She has been and will remain my primary source of inspiration in professional and personal life.

I dedicate this thesis to my parents and to my brother for their unwavering support and encouragement. My parents have been a great influence in my life. My mother showed me the joy of learning and my father taught me patience and perseverance, values that I find incredibly useful for research. I must thank my brother for his undying support. His faith in me is both humbling and inspiring.

Many people have generously provided financial support for my studies. I thank Venki, V.S. Sankar Narayanan, V.S. Sivakumar, V.S. Ravichandran, V.S. Prabhakar, Santosh Sivakumar, Barath Balasubramanian, Vikram Muthyam, Sreeram Veeraraghavan, Priya Krishnan, Shankarraman Nagesh, Srivathsan Krishnamohan and Santosh Perumbadi for their generosity towards me during difficult times.

I thank my friends. It is difficult to fully enumerate this group of incredible people.To those who have known me from before, thank you for never taking me seriously, and for enduring me these five years. To those who were my colleagues, thanks for making graduate studies fun. I owe my sanity to you.

I thank my advisor James Allan for all that he has done for me over the years. I thank him for encouraging me to pursue research that interested me, even if that meant working in areas that were outside of his research interests. I am grateful for the opportunity I have had to both work with him and to learn from him.

I thank my committee members, Bruce W. Croft, David Jensen and Michael Lavine. Their guidance and suggestions have greatly improved the quality of this thesis both in terms of research and presentation.

I thank my colleagues and collaborators over the years for the invaluable discussions, guidance and insights that have had a large influence in shaping my research. I thank Giridhar Kumaran, and Vitor Carvalho for their guidance during my internship in Microsoft and for their valuable mentorship since. I thank Silviu Cucerzan for his guidance during my internship in Microsoft Research and for his unstinting support and encouragement through these years. I thank Michael Bendersky for all his contributions and valuable discussions during our collaboration over these years.

I thank my colleagues at CIIR for all the stimulating discussion over these years. I have learned the most from them. I thank David Fisher for helping me every single time I went to him with a problem. I thank Dan Parker, Andre Gauthier, Kate Moruzzi, Jean Joyce and Glen Stowell for taking care of things big and small and for making my stay at CIIR wonderful.

I thank all the staff of the Computer Science department, who take exceptional care of their graduate students.

I thank the agencies that funded my research. This work was supported in part by the Center for Intelligent Information Retrieval, in part by the Defense Advanced Research Projects Agency (DARPA) under contract #HR0011-06-C-0023 and in part by NSF grant #IIS-0910884. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect those of the sponsor.

ABSTRACT

QUERY-DEPENDENT SELECTION OF RETRIEVAL ALTERNATIVES

SEPTEMBER 2011

NIRANJAN BALASUBRAMANIAN

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor James Allan

The main goal of this thesis is to investigate query-dependent selection of *retrieval alternatives* for Information Retrieval (IR) systems. Retrieval alternatives include choices in representing queries (query representations), and choices in methods used for scoring documents. For example, an IR system can represent a user query without any modification, automatically expand it to include more terms, or reduce it by dropping some terms.

The main motivation for this work is that no single query representation or retrieval model performs the best for all queries. This suggests that selecting the best representation or retrieval model for each query can yield improved performance.

The key research question in selecting between alternatives is how to estimate the performance of the different alternatives. We treat query dependent selection as a general problem of selecting between the result sets of different alternatives. We develop a relative effectiveness estimation technique using retrieval-based features

and a learning formulation that directly predict differences between the results sets. The main idea behind this technique is to aggregate the scores and features used for retrieval (retrieval-based features) as evidence towards the effectiveness of the results set.

We apply this general technique to select between alternatives reduced versions for long queries and to combine multiple ranking algorithms. Then, we investigate the extension of query-dependent selection under specific efficiency constraints. Specifically, we consider the black-box meta-search scenario, where querying all available search engines can be expensive and the features and scores used by the search engines are not available. We develop easy-to-compute features based on the results page alone to predict when querying an alternate search engine can be useful. Finally, we present an analysis of selection performance to better understand when query-dependent selection can be useful.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

The main goal of this thesis is to enable query-dependent selection of retrieval alternatives such as query representations and ranking algorithms. To this end this thesis presents:

1. A general technique for estimating the effectiveness of retrieval alternatives.
2. Application of this technique for two types of retrieval alternatives.
3. Extension of query-dependent selection under certain efficiency considerations.

In this chapter, we first motivate the need for query-dependent selection of alternatives and introduce the main components of this thesis.

1.1 Motivation

The main premise behind this thesis is that a query-dependent selection of retrieval alternatives can generalize better than a fixed selection across all queries.

Information retrieval systems convert a user input query into an intermediate *query representation*, which is then fed into a *retrieval model*. The retrieval model uses the query representation to score the documents in the collection, which are then presented to the user as a ranked list. The questions of which query representation to use, and which retrieval model to use are decided *a priori*, typically, using an average effectiveness metric on a set of training queries. This approach usually yields reliable *average effectiveness* on future test queries.

However, no single query representation or retrieval model performs the best for all queries (Croft, 1981; Bartell et al., 1994; Lee, 1995). This is because the effectiveness

of query representations and retrieval models vary for different queries. We use two examples to illustrate this point.

Consider the long query *easter egg hunts in columbus parks and recreation centers* that retrieves poor results on the Microsoft Bing search engine[1]. Removing the terms *recreation* and *centers* from the query yields substantial improvements in the search results. On the other hand, for shorter queries (less than four words) such as *computer science* removing any term is detrimental because each term is critical to the query's retrieval performance. Thus, for some queries reduced representations (with terms removed) are more effective, whereas for others the original representation is more effective.

As another example, consider two retrieval models R_{URL} and R_{body}. R_{URL} favors web pages that contain query terms in the URL, whereas R_{body} favors documents that contain the query terms in the body of the web pages. For the navigational query[2] *bank of america*, which can be used to find the home page of the institution Bank of America, R_{URL} is more effective than R_{body}. This is because, the homepage `www.bankofamerica.com` contains less than five mentions of the query terms *bank, of, america*, while there are several pages on the Web (especially news articles) that contain more mentions of the query terms. On the other hand, for the informational query[3] *colorado*, R_{body} is more effective. There are more than six million web pages that contain the term *colorado* in their URL[4], which suggests that the presence of keywords in URL alone is not adequate to rank documents for this query.

---

[1]Results obtained from www.bing.com as of January 2010.

[2]Navigational queries are those that are used to find a specific web page.

[3]Informational queries are not targeted towards a particular web page, but seek pages that provide information on the query's topic.

[4]According to Google's search results in March 2010.

These examples motivate query dependent selection: selecting the best representation or retrieval model for each query can provide substantial improvements over a single fixed choice for all queries. Our experiments on selecting between query representations and retrieval models show a large potential for query-dependent selection. For selecting between different reduced versions of long queries, we find that selecting the best reduced version can yield more than 30% relative improvements in NDCG@5, a rank-based metric used to evaluate the top 5 results returned by Web search engines (Chapter 2). Similarly, for selecting between two retrieval models we find there is a potential for more than 50% relative improvements in the fraction of relevant documents retrieved in the top 10 ranks.

1.2 Query-dependent Selection

We cast query-dependent selection as a general problem of selecting between result sets. To solve this results set selection problem, we propose a new effectiveness estimation approach, ReEff, which is based on retrieval features and learning formulations that can predict the differences between result sets. Figure 1.1 illustrates the application of query-dependent selection for selecting between retrieval models. We retrieve results using the available retrieval models (Model_0, ... , Model_m). We then apply ReEff to select the best result set to present to the user. The same approach is also useful for selecting multiple result sets for fusion – the technique of merging multiple result sets.

This thesis is organized around the following research questions:

1. How can we estimate the relative effectiveness of retrieval alternatives?

2. How to select between alternatives in query representations, and ranking algorithms?

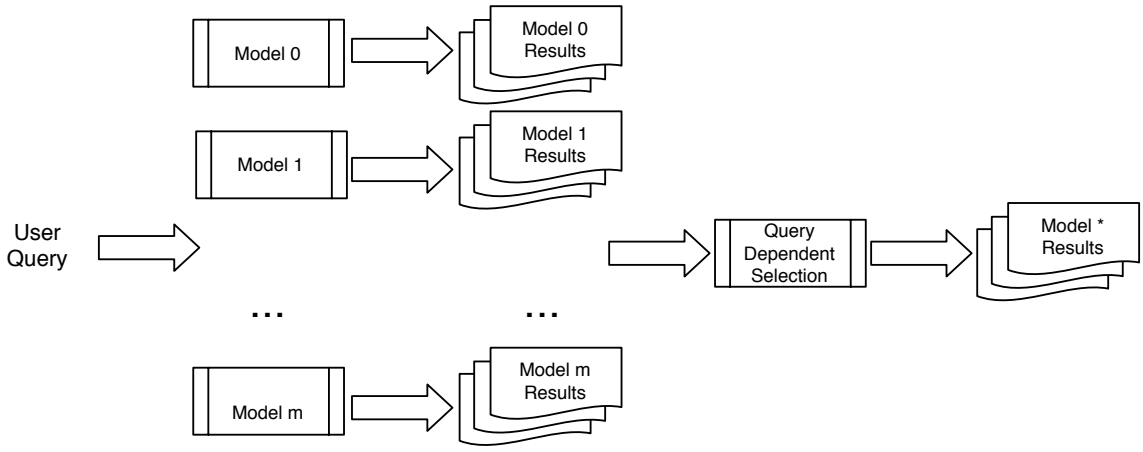


Figure 1.1: Query-dependent selection of retrieval models. User query is fed to multiple retrieval models, which produce corresponding result sets. The problem then becomes one of selecting the best result set for each query.

3. How to extend query-dependent selection of search engines under efficiency constraints in a black-box meta-search scenario?

4. When is query-dependent selection useful? In particular, what types of differences can be reliably detected, and what types of queries benefit from query-dependent selection?

1) ReEff - (Chapter 3)

The main challenge in selecting between result sets is estimating the effectiveness of the result sets. To address this challenge, we develop result set based features, which includes retrieval scores and aggregates of features, which are used to retrieve documents in the first place. Further, in addition to estimating the individual (absolute) effectiveness of each result set, we also focus on relative effectiveness estimation – that is, estimating the difference in effectiveness between the result sets. The rationale behind this approach is that accurate estimation of the absolute effectiveness value of each alternative is hard and not necessary. The estimated effectiveness values are useful only to the extent which they induce the right ordering among the alternatives and estimating the differences between the alternatives is closer to the

end goal of selecting an alternative.

2) Applications - (Chapters 4 and 5)

Our work shows that this approach can be successfully adapted for selecting reduced representations for long queries (Chapter 4), and for leveraging multiple ranking algorithms (Chapter 5). For choosing between reduced representations we utilize additional query-based features, whereas for choosing between ranking algorithms we incorporate additional result-set based features.

3) Efficiency Constraint - (Chapter 6)

We consider a specific efficiency constraint where querying all available alternatives for all queries is expensive. In particular, we focus on the black-box meta-search situation where a) querying all available search engines (or ranking algorithms) is expensive, and b) the underlying retrieval features or ranking scores are unavailable. In this scenario, we consider the problem of predicting when querying an alternate search engine can be useful. We develop URL and snippet based features and propose a simple combined measure that we directly optimize. Further, we develop a simple transfer learning approach that leverages easy to obtain overlap data to augment the training data to further improve performance.

4) Conclusions - (Chapter 7)

To conclude, we summarize the main findings of the thesis, discuss the limitations in terms of the insensitivity to small differences, the need for large amounts of training data and efficiency considerations. We also briefly discuss some lessons learned in this thesis that can be useful for future applications of query-dependent selection, and point to future work that are natural extensions to this thesis.

1.3 Contributions

To summarize, the main goal of this thesis proposal is to develop techniques for comparing the retrieval alternatives based on their estimated effectiveness. The main contributions of this thesis are the following:

1. ReEff – We develop a relative effectiveness estimation framework that allows us to choose between different result sets. This technique is particularly suited for Web search and works well when learning from large collections using multiple features. Experiments on a large web collection show that for Web search, the retrieval-based features used in ReEff are better predictors of result set effectiveness than Clarity, a content-based effectiveness predictor: ReEff's prediction achieve up to 0.78 in terms of Pearson's linear correlation. Our analysis shows that prediction quality varies for different effectiveness regions and that predicting performance for poor queries is more reliable than for easy queries. The initial results of this work, described in Chapter 3, were published as a poster paper in SIGIR 2010 (Balasubramanian et al., 2010b).

2. Query Reduction – We apply ReEff to the problem of selecting between reduced versions of long web queries. Our experiments on large web collection show that ReEff yields substantial improvements over using the original query alone: ReEff achieves about 4% average relative improvements on 25% of the queries, and more than 25% relative improvements on a smaller subset (5%) of long web queries. Further, we show that ReEff delivers most improvements to poorly performing queries, i.e. for queries which need the improvements most. The main results of this work, described in Chapter 4, were published as a paper in SIGIR 2010 (Balasubramanian et al., 2010a), extending prior work on automatic query reduction by Kumaran & Carvalho (2009).

3. Retrieval Models – Experiments on selecting between ranking algorithms show that ReEff can be used to select the best ranking algorithm for each query (3.75% relative improvement), as well as for combining multiple ranking algorithms through fusion (more than 4% relative improvement). Our analysis shows that ReEff can provide different types of benefits compared to fusion and that ReEff provides a nice trade-off between the number of queries impacted versus average performance: ReEff yields relative improvements of more than 13% on a smaller subset (around 5% of the queries) with a positive impact for more than 70% on this subset. The core idea of this work, described in Chapter 5, and some preliminary results (not included in this thesis) were originally published as a poster paper in SIGIR 2010 (Balasubramanian & Allan, 2010).

4. Efficiency – We develop a threshold-based classifier, LTI, for querying alternate search engines only when necessary. Easy-to-compute features based on the results page alone show promise for predicting when querying an alternate search engine can be useful. LTI achieves a 7% improvement over a competitive classification baseline. We develop a technique for automatically generating surrogate training data to further improve prediction accuracy by more than 15% over the classification baseline. The main idea behind this work and the results, described in Chapter 6, was first presented as a poster paper in the NESCAI 2010 student colloquium (Balasubramanian et al., 2010).

CHAPTER 2

BACKGROUND AND RELATED WORK

2.1 Background

In this section we present background material to introduce some of the alternatives in the design of an IR system and the basic approach for selecting alternatives.

A typical information retrieval (IR) system responds to a input query by generating a ranked list of documents. The input query is first converted to an intermediate *query representation*, which is then fed to a *ranking algorithm* that assigns scores to each document in the collection.

Query Representations - There are several ways to represent queries and documents in an IR system. Typically, a query is represented *as is* – using exactly the words specified in the query.

Alternatively, for short keyword queries automatically expanding the queries to include additional terms can help overcome vocabulary mismatch between queries and documents thereby improving retrieval performance. On the other hand, for long queries building a reduced representation by dropping some words can help improve retrieval performance. Determining which terms to drop is not trivial and for some queries the original representation itself might be the most effective.

Ranking Algorithm - A ranking algorithm is a function that assigns retrieval scores to documents. These retrieval (or ranking) scores can be viewed as estimates of relevance of each document given the input query. The documents are then presented to

the user in decreasing order of their scores in order to maximize retrieval effectiveness in accordance with the probability ranking principle (Robertson, 1977)[1].

The ranking algorithms perform a weighted combination of retrieval features to produce retrieval scores. Retrieval models are a class of ranking algorithms that combine a small number of features. TF-IDF (Spärck-Jones, 1972), Okapi BM25 (Robertson et al., 1994) are retrieval models that combine two types of term-level features, term frequency (TF)and inverse document frequency (IDF).

Learning-to-rank algorithms are another class of ranking algorithms that are used when combining large number of retrieval features. For example, ranking algorithms for web search include a large number of features that include a) term level features such as TF, IDF, b) scores from the retrieval models themselves, c) document quality features such as page rank, number of inlinks and outlinks, and d) click-through history based features such as click counts of query-url pairs. The weights for combining these different features are learned using machine learning techniques such as RankSVM (Joachims, 2002), Co-ordinate Ascent (Metzler, 2007), and AdaRank (Xu & Li, 2007). Despite the differences in the techniques used for training, the output of each of these learning algorithms is a set of weights that are used to produce a weighted combination of the retrieval features.

The design of an IR system involves selection of suitable query representations and ranking algorithms. The basic approach behind IR system design is to evaluate the available alternatives (of each type) on a set of queries for which relevance information is available i.e., on queries for which the relevant documents are known[2]. Given relevance information, the performance of the alternatives for each query can be evaluated using *effectiveness measures*. The performance of an alternative is measured

[1]The maximization is guaranteed only under the assumption that the relevance of a document (or its utility) is independent of the other documents present in the ranked list.

[2]In practice, all relevant documents are not known. Therefore, evaluations are often based on incomplete relevance information.

by the effectiveness of the result set retrieved when using the alternative. Example effectiveness measures include 1) average precision, which is the average of precision at ranks where relevant documents are retrieved, 2) discounted-cumulative gain (DCG), which is a measure that handles graded relevance and discounts the contribution of documents lower in the ranked list (Järvelin & Kekäläinen, 2002), and 3) normalized discounted-cumulative gain (NDCG), which is DCG normalized based on the number of relevant documents that exist for a given query (Järvelin & Kekäläinen, 2002).

Typically, the effectiveness measures averaged over many queries are used to select the best performing alternatives. The selected alternatives are fixed for all future queries. Instead, we seek to select the best alternatives for each query by estimating the effectiveness of each alternative for the given query. In particular, we focus on selecting reduced representations for long queries (query reduction) and selecting the best ranking algorithm for Web search.

Below, we describe prior work in estimating effectiveness, and its application to selecting between alternatives in query representations and ranking algorithms. We also describe prior investigations into efficiency versus effectiveness trade-offs in querying multiple search engines.

2.2 Effectiveness Estimation

The task of estimating retrieval effectiveness has been widely studied (Cronen-Townsend et al., 2002; Hauff et al., 2008; He et al., 2008; Zhao et al., 2008). Effectiveness estimation techniques can be broadly grouped into two categories: 1) *Pre-retrieval* - estimation before performing retrieval using query-based features alone, 2) *Post-retrieval* - estimation based on properties of the retrieved results.

Estimation techniques are useful when they are both effective and efficient. Pre-retrieval techniques do not retrieve documents to estimate effectiveness. Instead, they use collection statistics of query terms such as inverse document frequency, collection

term frequency, and variance of term weights, to predict query performance (He & Ounis, 2004b; Amati et al., 2004; Zhao et al., 2008). Avoiding retrieval and analysis of the search results improves prediction efficiency but it also limits prediction effectiveness, as the quality of the search results depends heavily on the retrieval algorithm. Therefore, pre-retrieval techniques are often more efficient but less effective when compared to post-retrieval techniques.

Effective post-retrieval techniques are based on properties of the result set. One of the most effective post-retrieval measure is Clarity. Clarity measures the divergence of the language model[3] built from the retrieved documents from the language model of the background collection (Cronen-Townsend et al., 2002). Larger divergences correlate with higher query performance.

There are two main issues in applying clarity based approaches to web search. Clarity is not effective for web collections and involves expensive analysis of the content of the retrieved documents to build language models.

Zhou & Croft (2007) argue that the diversity of web queries, and the diversity of the retrieved documents lead to poor estimation. Hauff et al. (2008) further show that Clarity is sensitive to the number of documents and terms used to for building language models. Zhou & Croft (2007) propose query feedback, a rank sensitivity based measure, and weighted information gain as alternatives, whereas Hauff et al. (2008) propose an approach that dynamically chooses the number of documents and the terms used for computing Clarity. All of these approaches show improvements over Clarity on the TREC GOV2 web collection.

The query feedback approach views retrieval as a noisy channel model, which converts the original query Q to a noisy version $Q^{'}$. The overlap between the top ranked

[3]In the language modeling framework for retrieval, queries, documents, and collections of documents are typically represented as multinomial distributions specified over the vocabulary. The parameters of these distributions are estimated using a maximum likelihood estimation.

results of the original and the noisy version is then used as an estimate of the effectiveness. Computing the noisy query requires expensive analysis of retrieved documents, as well as conducting an additional retrieval. The rank-sensitivity approach utilizes an approach that is similar to the perturbation models used to measure the robustness of retrieval described by (Zhou & Croft, 2006), which involves multiple retrievals. Hauff et al. (2008)'s approach only limits the number of documents and terms used in building query models but still requires expensive analysis of the retrieved documents. Our experiments on a large collection of Web queries also shows that Clarity performs substantially worse compared to the less expensive features that we propose (Chapter 3).

The weighted information gain approach measures the relevance of each document with respect to the query using a MRF based model. The relevance estimates are normalized by the relevance of an average document to make them comparable across queries. This approach utilizes term proximity based features in an MRF model to estimate each document's relevance. Instead, we use a more general approach – compute multiple aggregates of different estimates of relevance, such as retrieval scores and retrieval features.

Other examples of post-retrieval techniques include measures of similarity among the retrieved documents (Zhao et al., 2008), measures of coherence of the top ranking documents (He et al., 2008), and robustness of the retrieval performance (Zhou & Croft, 2006), which require post-processing of the retrieved documents, or multiple retrievals.

In summary, effectiveness estimation using content-based techniques can be expensive and are not well suited for query-dependent selection on the Web.

2.2.1 Key Differences

Our approach to effectiveness estimation, ReEff, is driven by the need to select between alternatives in retrieval and its suitability for Web search. It differs from the previous approaches in two ways.

First, instead of computing content-based measures on the result sets, ReEff *directly* uses retrieval scores and features that are used by the retrieval algorithm itself. These features have two main advantages: 1) they are readily-availabe during retrieval, and 2) they directly affect the ranking of the documents and are more directly related to the effectiveness of the result set. Retrieval score-based features have been used to predict query performance on TREC collections but with mixed success (Piatko et al., 2004; Tomlinson, 2004). We show that these retrieval based features are strong predictors of effectiveness for Web search (Chatper 3).

Second, unlike prior work that focused on ranking different queries (or estimating the difficulty of individual queries), we focus on the setting where the effectiveness estimation is used for selecting between different ways of conducting retrieval for the same query. This implies that the effectiveness estimation is useful primarily in terms of the ordering it induces on the alternatives at hand. Therefore, we focus on modeling differences in effectiveness instead of independent estimation for each alternative. We find that the difference prediction approach is more useful than independent estimation for the two applications that we consider (Chapters 4 and 5).

2.3 Query-dependent Strategies

Early work on query-dependent application of search strategies by Croft & Thompson (1984), described an adaptive mechanism that utilized simple query-based features alone for selecting between two types of retrieval models. The limited performance was attributed to the difficulty of finding good performance predictors on the limited training data that was available. More recently, query-dependent application

of retrieval techniques have been studied in various contexts including query reduction (Kumaran & Carvalho, 2009), selective query expansion (Cronen-Townsend et al., 2004; Amati et al., 2004), query-dependent learning (Qin et al., 2007; Bian et al., 2010; Geng et al., 2008) and query-dependent selection of ranking functions (Plachouras et al., 2004; Peng et al., 2009).

2.3.1 Query Reduction

Query reduction is an approach that has been shown to have great potential for improving retrieval effectiveness of long queries (Kumaran & Allan, 2007). Kumaran & Carvalho (2009) develop an automatic method for reducing long TREC description queries. Using content-based effectiveness predictors such as Clarity and Mutual Information Gain, they convert the query reduction task into a problem of ranking (reduced) queries based on their predicted effectiveness. Their results on TREC Robust 2004 show the viability of automatic query reduction. In this work, we generalize the problem of query reduction to select between all reduced representations – the original long query, as well as the reduced versions. We develop adaptations that make query reduction more suitable for use in web search engines.

Lee et al. (2009) use statistical and linguistic features of query terms to greedily select query terms from the original long query to achieve the effect of query reduction. Experiments on NTCIR collections demonstrate that this approach, and its extension which considers pairs of terms (Lee et al., 2009) improves long queries' effectiveness. Chen & Zhang (2009) use personal query history to select short queries related to the original long query, cluster the short queries based on similarity of their contexts, and select representatives from each cluster as a substitution for the original long query.

These approaches either only utilize query-dependent features and do not utilize result set based features or require expensive rank-time processing of texts of retrieved documents to compute reduced versions.

2.3.2 Query expansion

Query expansion is an approach that aims to automatically add terms to the user input query (original query) to address vocabulary mismatch between the user input query and the relevant documents in the collection (Attar & Fraenkel, 1977; Croft & Harper, 1979; Efthimiadis & Biron, 1993; Buckley et al., 1995; Xu & Croft, 1996). Pseudo-relevance feedback, also known as local feedback or blind feedback, is a popular query expansion technique that extracts terms from the top ranked documents obtained using an initial retrieval with the original query. The extracted terms are then used to perform a second round of retrieval and the retrieved documents are then presented to the user (Attar & Fraenkel, 1977; Croft & Harper, 1979).

Using pseudo-relevance feedback often improves the average performance but its application is limited because it can lead to drastically worse results in some cases. To address this issue several modifications have been proposed. These approaches can be broadly grouped into a) selective expansion using predictive mechanisms such as query-drift (Cronen-Townsend et al., 2004; Amati et al., 2004), b) improving the estimation of expansion models by separating out influence of non-relevant documents (Tao & Zhai, 2006; Lee et al., 2008), and c) merging results from multiple expansion models (Collins-Thompson & Callan, 2007; Zighelnic & Kurland, 2008; Collins-Thompson, 2009).

Our approach is similar to these approaches in the general idea of using estimates of effectiveness to select the best representation. While query-dependent strategies for expansion have shown success, the techniques model aspects that are specific to query-expansion, which do not apply to query-reduction or selecting ranking algorithms.

For example, selective expansion relies on the notion that a poorly performing query will also perform poorly when expanded. Also, these techniques require expensive computations that involve analysis of the retrieved documents. These computations do not add substantial overhead relative to the cost of expansion itself. However, in the context of Web search these computations can be prohibitively expensive.

2.3.3 Ranking Algorithms

Developing retrieval models and ranking algorithms is one of the most active areas of research in IR, which has resulted in several successful retrieval models, and led to the development of several learning to rank techniques. Combining multiple sources of evidence for retrieval has been studied in various contexts (Croft, 1981; Turtle & Croft, 1991; Belkin et al., 1993; Bartell et al., 1994; Lee, 1995; Farah & Vanderpooten, 2007). In this section, we focus on combining evidence from multiple ranking algorithms or retrieval models. The basic premise for combining evidence from multiple retrieval models is that there is no *single* model that performs the best on *all* queries.

Methods for combining multiple retrieval models can be broadly categorized as *model selection* and *fusion* techniques.

Model selection is the problem of selecting the best retrieval model for each query. He & Ounis (2004a) use query-based pre-retrieval features to identify a cluster of queries in the training set, and select the best retrieval model on the cluster. Geng et al. (2008) utilize a similar approach but instead, identify the top-k nearest neighbors in the training set for a given test query and use these neighbor queries to train a ranking function. Zhu et al. (2009) propose grouping queries based on their difficulty using features such as click entropy and learn different ranking functions for each group. Peng et al. (2009) propose a more efficient and effective approach that utilizes the performance of the ranking algorithms on the neighbor queries to

select the best ranking function for the test query. This approach allows for the use of different ranking algorithms. However, as we show in Chapter 5, the use of neighbor queries to predict performance on test queries does not scale well for large collections.

Other works propose modifications to learning techniques. Qin et al. (2007) propose a modification to RankSVM that learns multiple hyperplanes (one for each top K rank), whose rankings are then combined to produce a single ranking. Bian et al. (2010) develop query dependent loss functions which adjust the contribution of each query to the learning, based on the query's type: whether it is informational or navigational.

Fusion techniques use rank fusion (Cormack et al., 2009; Montague & Aslam, 2002; Lillis et al., 2006; Aslam et al., 2005; Lee, 1997) and rank aggregation (Farah & Vanderpooten, 2007; Klementiev et al., 2009) to re-rank documents based on retrieval scores (or rankings) obtained from individual retrieval models. However, most of these approaches either learn a fixed (query independent) set of weights that are used to combine document scores or utilize a voting scheme for combining the rankings. Manmatha et al. (2001) use Gaussian and normal distributions to represent non-relevant and relevant score distributions to estimate the probabilities of relevance for each retrieval model, which are then used to improve fusion. ReEff on the other hand aims to directly model differences in effectiveness without modeling underlying relevant and non-relevant score distributions, while also using features that are used for retrieval in the first place.

2.3.4 Key Differences

The key difference of our approach with respect to the pre-retrieval based model selection approach (He & Ounis, 2004a) is that we utilize the rankings on the test query to select the ranking algorithm. The neighbor query finding approaches utilize test rankings but use them indirectly, either to train new ranking functions (Geng

et al., 2008; Zhu et al., 2009) or to predict performance based on average training performance (Peng et al., 2009). In contrast, we estimate performance based on the test rankings and also focus on the differences in performance, which leads to better generalization performance on test queries in large Web search collections.

We also show that query-dependent selection of rankers can help fusion. We perform selective fusion – fusion for some queries, selection for others, selecting rankers for fusion, and weighting rankers for fusion.

2.4 Efficiency constraints in Black-box Meta-Search

In the meta-search setting, multiple rankers can be used to either route the user to the most effective search engine (White et al., 2008) or fuse search results to produce a new ranking (Selberg & Etzioni, 1995). We consider the setting where accessing all rankers for all queries is expensive. Furthermore, we consider the black-box meta-search setting where the features and the scores used by the ranking algorithms are inaccessible to the meta-search engine. In this constrained meta-search scenario, predicting how many new relevant documents can be obtained from an alternate search engine is useful for both efficiency and effectiveness reasons.

White et al. (2008) develop query and result-set dependent techniques to solve the problem of automatically routing users to the search engine that provides the best result for a given query. We extend this problem to the black-box meta-search scenario with efficiency constraints.

Predicting relevance gain from an alternate search engine can also be helpful for fusion. Beitzel et al. (2003, 2004) show when the search engines have systemic differences[4], the improvements from fusion are largely due to the presence of different relevant documents in the search results rather than better re-ranking of the common

[4]Systemic differences include differences in tokenization, stemming, stop words etc., in addition to differences in the retrieval models themselves.

relevant documents alone. In the web meta-search scenario, where the rankers are both effective and diverse, rankers with low overlap in search results are more likely to produce better fused results compared to rankers with higher overlap (Dogpile.com, 2007).

The effectiveness gains versus the loss in efficiency when querying multiple resources (different document collections) has been studied in the context of federated search. The main goal is in selecting a small number of resources that maximize the number of relevant documents returned. Selecting too few resources might yield low recall, whereas selecting too many resources can be inefficient. Si & Callan (2005) utilize a centralized sample of documents created from past queries to estimate the relevance of search engines in order determine the smallest set of search engines that maximize the expected utility. Cetintas & Si (2007) propose an extension which considers the cost of downloading search results to determine the number of documents to download for results merging. Arguello et al. (2009) use several corpus dependent, query-category based, and click-based features to select few sources (collections) whose results can be combined with effectiveness comparable to a full retrieval on all collections.

Baeza-Yates et al. (2009) investigate the effectiveness versus efficiency issues in a two-tiered Web search model with a local server, and a remote server. Usually, every query is routed to a local server first, and is routed to the secondary server only if the results from the local server are deemed inadequate. To avoid the poor response times due to sequential querying, they propose an approach that is able to predict whether the local server's results will be sufficient prior to retrieval.

2.4.1 Key Differences

Our work differs from these federated search approaches both in terms of the additional constraints imposed by the *black box* metasearch scenario and the techniques

employed. First, in contrast to the standard federated search setting (Arguello et al., 2009; Baeza-Yates et al., 2009; Cetintas & Si, 2007), each query to a ranker incurs a cost regardless of the number of top-k search results obtained, and the meta-search engine has no direct access to full document texts and rankers indexes. In addition, instead of solely relying on inter-ranker overlap, as is done in some previous work (Baeza-Yates et al., 2009), we utilize a relevance gain metric when relevance judgments are available. Finally, we develop a technique to automatically create surrogate training data in cases when relevance judgments are scarce.

Summary

This thesis focuses on the problem of query-dependent selection and seeks to develop a general approach for selecting between retrieval alternatives. We describe the application of this approach for two types of retrieval alternatives. It differs from prior work in terms of the features used for estimating effectiveness, and the use of learning formulations that directly estimate relative effectiveness instead of absolute effectiveness. Further, we investigate the efficiency versus effectiveness trade-offs in a specific meta-search setting. In contrast to prior work, we consider additional constraints on the available features and target the prediction of the amount of new relevance information when querying alternate search engines.

CHAPTER 3

RELATIVE EFFECTIVENESS ESTIMATION

In this chapter, we describe ReEff, our technique for query-dependent selection of retrieval alternatives. First, we describe the features we use to estimate effectiveness. Then, we present effectiveness estimation experiments on a large collection of web search queries. Finally, we describe the learning formulations that we use to combine these features for selecting retrieval alternatives.

In the chapters that follow (Chapters 4 and 5), we describe the application of this technique to select between reduced versions of long queries, and to leverage multiple ranking algorithms.

3.1 Approach

The main idea behind ReEff is to use the scores of the top ranked documents of alternatives, and the features used by the retrieval algorithms for these top ranked documents, and combine them to produce a single measure for selecting between these alternatives.

We use a simple example of selecting between two retrieval models to illustrate the main idea. Retrieval models are functions that assign retrieval scores to each document in the collection $Score : \mathbf{Q} \times \mathbf{D} \rightarrow \mathbb{R}$. The score for each document is computed as a weighted combination of retrieval features. The values of the retrieval features are computed by feature functions that are defined over query-document pairs. Examples include features such as as term frequency (TF), inverse document frequency (IDF), and page rank. The score of a document d with respect to a query

q is given by $Score(q,d) = \sum_{i=1}^{i=n} w_i \cdot f_i(q,d)$, where w_i refers to the weight of the feature function f_i.

Retrieval models can differ in the set of feature functions they use, as well as the weights they use to combine the values of the feature functions. For simplicity let us consider the case where we have two retrieval models A and B that use the same set of feature functions but differ in the weights that they use to combine these feature functions. Figure 3.1 shows the basic idea behind ReEff for selecting between the two retrieval models A and B. We first rank the documents in the collection using the model A and then select the top K documents in this ranking. We collect the scores assigned to these top K documents ($Score$ 1, $Score$ 2, $\cdots$, $Score$ k). We also extract the retrieval features from these top k documents. For each feature function f_i, we aggregate its values for the top K documents ($f_{i1}, f_{i2}, ..., f_{ik}$), where f_{ij} denotes the value of feature function i for document j i.e., the value of the i^{th} retrieval feature in the j^{th} document.

Then, for both the retrieval scores and retrieval features we compute statistical aggregates such as min, max, mean, standard deviation. In Figure 3.1, we refer to these aggregate features as $(a_0^1, a_0^2, ..., a_0^m, a_1^1, a_1^2, ..., a_1^m, a_2^1, ..., a_n^m)$[1] for model A, where a_{ij} refers to the jth aggregate of feature type i.

The procedure is repeated for model B. Together these aggregate features are then used in a learning formulation to select the best model.

Section 3.2 discusses the intuitions behind these features and presents empirical evidence that supports the use of these features for effectiveness estimation. Section 3.3 describes the learning formulations that we investigate to combine these features for query-dependent selection.

---

[1]We use a_0^j to refer to the jth aggregate of retrieval scores.

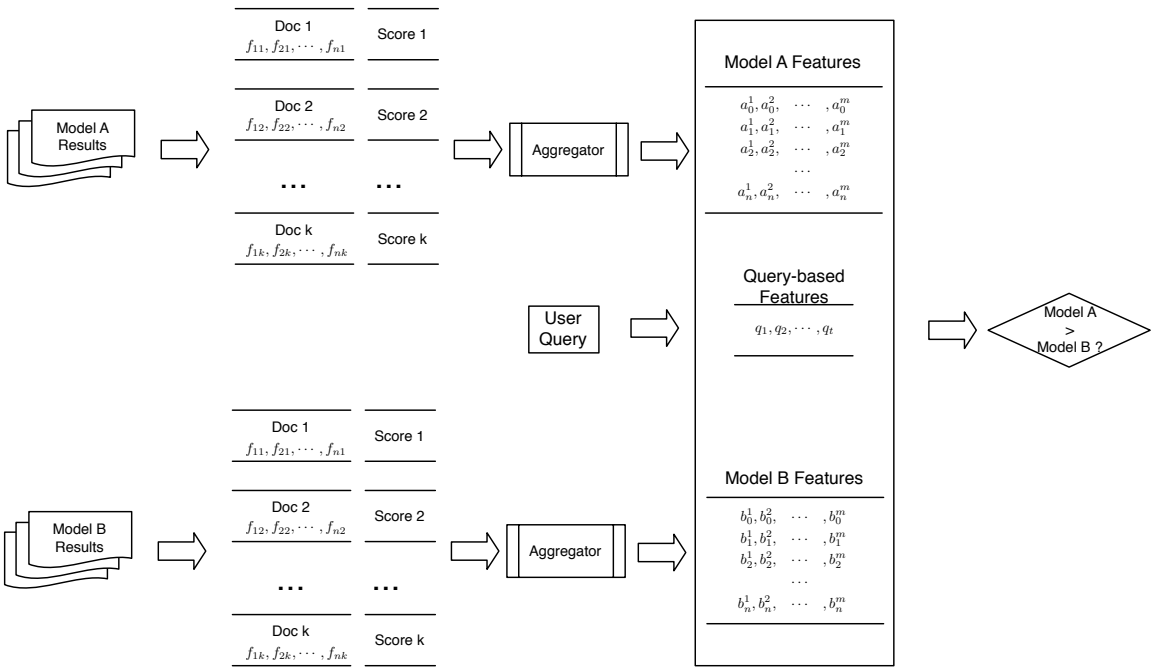


Figure 3.1: The Relative Effectiveness Estimation (ReEff) technique for query-dependent selection between two retrieval models A and B.

3.2 Features

The two applications we focus in this thesis (query reduction and ranker selection) are related to Web search. Therefore, we seek to develop effectiveness estimation features that are suited for the Web. First, we want the features to be reliable predictors of effectiveness for Web queries. Second, we want the features to be easy-to-compute during run-time to accommodate the milli-second latencies imposed by web search engines. To satisfy these goals we leverage the retrieval scores and features that are already used by the ranking algorithms.

The retrieval scores of the top-ranked documents can be viewed as estimates of relevance and thus directly relate to effectiveness. Intuitively, if retrieval scores at the top-ranked positions of the result set are higher, then the effectiveness of the result set is likely to be higher. We use the scores of the top-ranked documents at each position as independent features. Absolute values of the retrieval scores are not

directly comparable across queries (Larkey & Croft, 1996), we use a simple min-max normalization to scale all retrieval scores to a common range.

Different aspects of the score distribution can reflect different aspects of the quality of the results set. Large values for the mean of top-ranked documents tends to indicate higher effectiveness, whereas large variance in the scores can be an indicator for a less cohesive results set suggesting lower effectiveness. Indeed, we find that mean is positively correlated with effectiveness, whereas variance is negatively correlated with effectiveness. To capture these different aspects of results set quality, we use multiple statistical aggregates such as mean, variance, and maximum.

Similarly, the features used to produce the retrieval scores are also intimately related to retrieval effectiveness as they are selected to provide evidence towards the relevance of the documents. For example, the scores of retrieval models such as BM25, TF-IDF etc are often used as retrieval features for web search. These can be viewed as different estimates of document relevance that can be aggregated to reflect the quality of the result set.

Note that the scores and retrieval features are readily available during run-time. Computing simple aggregates of these features over the top few ranked documents adds a relatively small computation overhead, compared to techniques that require an analysis of retrieved documents. As we will show in the following section these features are strong predictors of retrieval performance for Web search and are also more suitable for Web search from an efficiency standpoint.

We present additional query-based features for the query reduction application in Chapter 4 and additional result-set based features for leveraging multiple rankers in Chapter 5.

Table 3.1: Regression features used for performance prediction. Pos. indicates position based features, where the value for each top-k document is used as an independent feature. Agg. indicates that statistical aggregates (mean, max, standard deviation, variance and coefficient of dispersion) of the values for the top-k documents are also used. For each feature, we perform a min-max normalization to rescale values between $[0, 1]$.

Feature Name	Description	Variants
LR	LambdaRank score of top-k documents	Pos. and Agg.
BM25	Okapi-BM25 score of top-k documents	Pos. and Agg.
Click-based	Click-through counts and other variants	Agg.
Static Scores	Page-rank like scores of the top-k documents	Agg.

3.2.1 Experiments

We conduct prediction experiments to evaluate the utility of the retrieval-based features for estimating the effectiveness of web search results. We target the predicting the effectiveness of web search results in terms of DCG@5[2] and NDCG@5.

We use a large proprietary collection of 12,185 queries that were sampled from the query logs of Microsoft Bing[3]. For each query in this collection, we create feature vectors as follows. First, we use LambdaRank to assign scores and rank documents. Our implementation uses several retrieval features such as BM25 (Robertson et al., 1994) features of different fields in the documents, click-based features such as query-url click count and its variants, query length, and other query independent features such as static rank (see Appendix for a partial but more detailed listing of these retrieval features).

For each of these retrieval features we create statistical aggregates. Next, we select the top 100 aggregates that have the highest linear correlation with the target metric on a set of training queries. Table 3.1 displays the list of features used for regression. We refer to these features as regression features henceforth. Finally, we

[2]We normalize DCG@5 by the perfect DCG@5 to scale values to (0,1).

[3]`http://www.bing.com`

create a query performance prediction dataset by associating with each query, the performance metric, DCG@5 or NDCG@5 and the regression features. Using this dataset, we conduct three-fold cross-validation experiments to train linear regressors, as well as a non-linear regressor based on the Random Forest algorithm (Liaw & Wiener, 2002). We used the *randomForest* package available from R with default parameters[4].

3.2.2 Results

Table 3.2 shows the prediction accuracy for DCG@5 and NDCG@5 [5] in terms of linear correlation and root mean squared error (RMSE) of the predicted and actual DCG and NDCG values.

Both predicted DCG and predicted NDCG values achieve a high linear correlation and low RMSE. Also, NDCG prediction is much harder as indicated by the low correlation and higher RMSE values. This is mainly because NDCG is a non-linear metric that is computed based on the actual number of relevant documents that exist in the collection. This information cannot be estimated based on the features of the top ranked documents alone. The overall prediction accuracies of simple linear regression and the non-linear random forest based regression are similar in terms of both metrics.

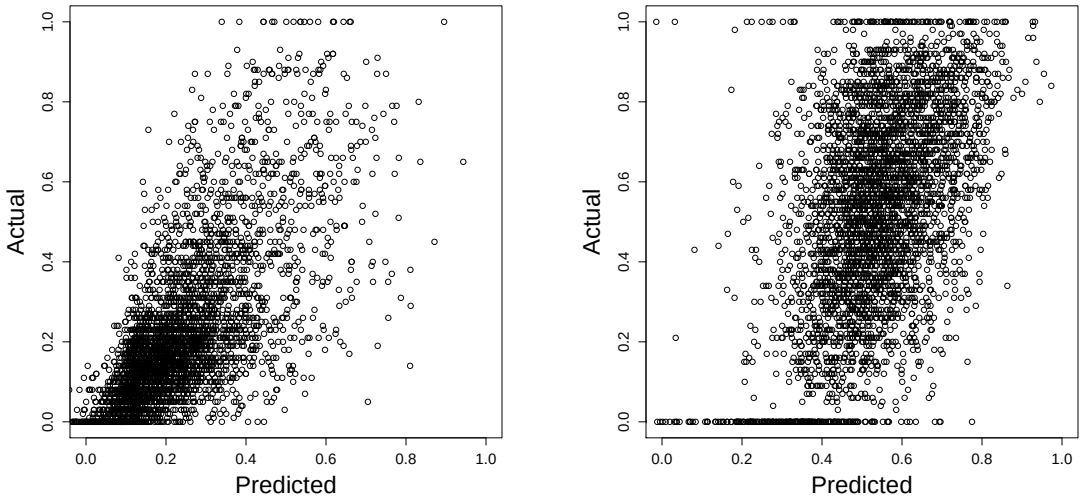
Table 3.2: DCG and NDCG Prediction accuracy of linear and non-linear regression.

Method	DCG		NDCG	
	Correlation	RMSE	Correlation	RMSE
Linear	0.78	0.13	0.50	0.23
Random Forest	0.79	0.13	0.52	0.22

[4] `http://cran.r-project.org/web/packages/randomForest/index.html`

[5] For the remainder of this chapter, we use DCG and NDCG to refer to DCG@5 and NDCG@5 for brevity.

The scatter plot in Figure 3.2(a) illustrate a strong correlation between the predicted and actual DCG values for a single fold. Figure 3.2(b) shows predicted NDCG values which are not as strongly correlated with the actual values. For DCG, when the predicted values are less than 0.2, the actual values are also less than 0.2 in most cases. On the other hand, when the predicted values are greater than 0.4 the actual values are more spread out. This suggests that DCG prediction is more precise for *hard* queries than for *average* and *easy* queries. Similarly, NDCG prediction is highly precise when predicted values are below 0.3. However, prediction effectiveness degrades quickly when predicted values are greater than 0.4. Thus, for both DCG and NDCG, the high linear correlation and low RMSE values mask the rather poor effectiveness at the extremes.



(a) DCG (b) NDCG

Figure 3.2: Linear Regression: Prediction versus Target Metrics for Test fold 1.

Feature Importance. We inspect the features used for the DCG and NDCG regression. Note that the features selected for DCG and NDCG can be different. We consider three subsets: features based on 1) *LambdaRank* scores, 2) *Click*-based features, and 3) *BM25F*-based features.

Table 3.3: Prediction Effectiveness of different feature groups.

Group	DCG		NDCG	
	Correlation	RMSE	Correlation	RMSE
LambdaRank	0.75	0.14	0.50	0.22
Click	0.78	0.13	0.41	0.24
BM25F	0.71	0.14	0.38	0.24
All	0.78	0.13	0.50	0.23

Table 3.3 shows the prediction effectiveness of the different feature groups for linear regression. For DCG, all feature groups achieve high correlation while for NDCG, click and BM25F features are substantially lower compared to the combined features. Also, relative feature importance differs for DCG and NDCG. For instance, click features are more important for predicting DCG than LambdaRank features while, the relationship is reversed for NDCG. Click features are strong predictors of user preference (Agichtein et al., 2006; Carterette & Jones, 2008), and it is no surprise that they correlate well with DCG. However, NDCG, being a non-linear metric, is harder to predict with click-based features alone. We hypothesize that since LambdaRank combines several features including click features and is trained to optimize for NDCG, the LambdaRank-based features turn out to be better predictors than click features. It is also interesting to note that the click features for DCG and LambdaRank features for NDCG are as effective as all the features combined. This suggests that more careful feature selection can reduce the run-time computations while retaining prediction effectiveness.

For comparison purposes, we also show the performance of Clarity (Cronen-Townsend et al., 2002). Our implementation of Clarity uses a query model built from the top 100 results returned by the search engine. We build the query models from query-biased snippets rather than from the entire text of the documents. In addition to being efficient, we find that it also helps create better quality query models by focusing on the relevant portion of the web pages and automatically filtering out layout information and advertisements. When compared to the features we use, clar-

ity achieves very low correlation for both DCG and NDCG, as shown in Table 3.4. The poor performance on this large web search collection is similar to the results observed on smaller TREC Web collections (Zhou & Croft, 2007).

Table 3.4: Correlation: *Average*, *Best* and *Worst* correspond to the average feature correlation, the highest and lowest correlation of the features used in our approach.

Feature	NDCG correlation	DCG correlation
Clarity	0.16	0.09
Worst ReEff	0.20	0.17
Average ReEff	0.57	0.27
Best ReEff	0.70	0.50

3.2.3 Gov2 Experiments

We also conduct experiments on small scale collection of 150 queries from the TREC Gov2 collection. We compare ReEff's performance with Clarity and Weighted Information Gain (WIG) – a measure that was shown to perform better than Clarity on the TREC Gov2 collection (Zhou & Croft, 2007). The main purpose of these experiments is to characterize the performance of ReEff when learning from small amounts of data using a small number of features.

Following Zhou & Croft (2007), we report results on the two partitions of this data set, TB04+05 and TB06. They use Sequential Dependence Model (SDM) (Metzler & Croft, 2005) to rank documents and aggregate the scores to compute WIG. For comparison purposes, we also use SDM to perform retrieval using Indri search engine and the same parameters that they use. We then compute ReEff's features using the single SDM based ranking scores.

Table 3.5 shows the linear correlation with average precision when using Clarity, WIG, and ReEff for estimating the average precision of each query. We find that ReEff's features are not as effective on this smaller collection. While the overall

performance is better than Clarity, it is worse compared to WIG. While there are changes in the Indri implementation and the choice of stop words, we attribute the large differences to methodological differences. There are two key differences in this setting that can contribute to the relative poor performance of ReEff.

Table 3.5: Linear Correlation with Average Precision on two partitions of the TREC Gov2 collection – TB04+05 and TB06. The techniques are trained on one partition and tested on the other.

Collection	Clarity	WIG	ReEff
TB04+05	0.33	0.57	0.40
TB06	0.08	0.46	0.30

First, the amount of training data available in this setting is small. When learning from small number of training examples, we hypothesize that normalizing relevance estimates across queries is more important and as a result ReEff which utilizes simpler un-normalized aggregates does not perform as well. For example, WIG uses normalization of the relevance estimates with respect to an average document in the collection. Further, as noted by Zhou & Croft (2007), the sequential dependence model parameters are also adjusted to perform a length-based normalization role in the case of generating estimates.

Second, the performance of ReEff is also related to the use of multiple estimates of relevance. In this setting, we only use the un-normalized aggregates of SDM scores as features, while in the larger web collection, we had access to multiple retrieval-based features. To see the impact of adding more features, we used BM25 scores for the top-ranked documents to compute additional aggregates and we saw improvements in linear correlation for both partitions (0.51 and 0.38 respectively).

In summary, we find ReEff to be more useful for settings where we have access to large amounts of training data and multiple estimates of individual document relevance. The poor performance of ReEff compared to WIG suggests that normalization

of documents scores is important when learning to estimate effectiveness from small collections.

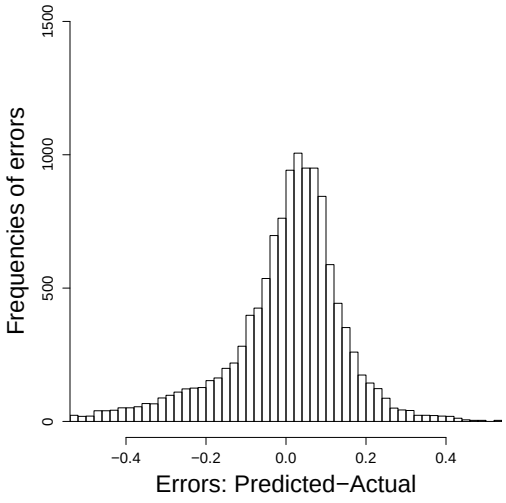
3.2.4 Analysis

In this section, we further analyze of the effectiveness of our prediction techniques on the large scale data set. Unless otherwise stated, all analyses are based on the results of linear regression using the top 100 highly correlated features.

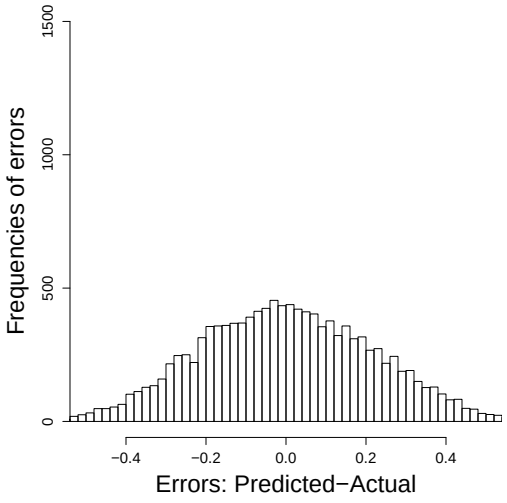
Distribution of Errors. The distribution of DCG prediction errors is concentrated around zero as shown in Figure 3.3(a). In fact, around 80% of the errors are within 0.2 of the actual values. However, the errors span a relatively high range of values compared to the true distribution of DCG, shown in Figure 3.3(c). Figures 3.3(b) and (d) show corresponding distributions for NDCG. Nearly 80% of the errors are within 0.4 of the actual values. NDCG errors are more spread out compared to DCG errors because NDCG prediction is harder. Also, NDCG values are more evenly distributed, thus increasing the range of possible errors.

Figure 3.3(e) shows that 1) most of the prediction errors are small for hard queries (shown in the left portion of the plot) and 2) errors increase for easy queries (shown in the right portion of the plot) . For example, for queries with DCG$<$ 0.1 (hard queries) most errors are less than 0.1 but for queries with DCG$>$ 0.5 (easy queries), most errors are above 0.4. On the other hand, NDCG errors are higher at both extremes, as shown in Figure 3.3(f). The distribution of errors show that despite the high values for the average metrics, prediction effectiveness for *hard* and *easy* queries needs further improvement.

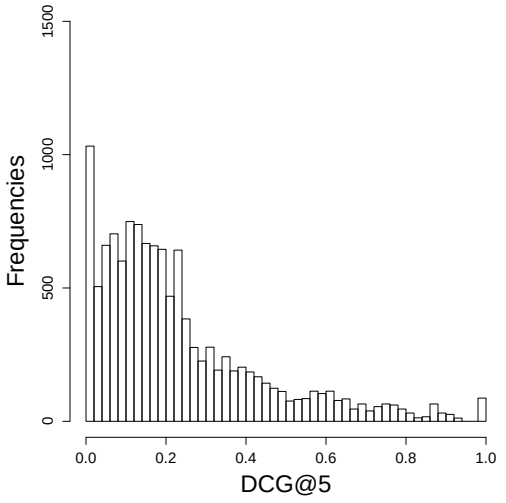
Effectiveness Regions. Even though metrics such as linear correlation and RMSE are useful for training and evaluation, from an application standpoint these average metrics are not adequate. For example, reliable identification of *easy* queries is useful for selective application of pseudo-relevance feedback (Amati et al., 2004)



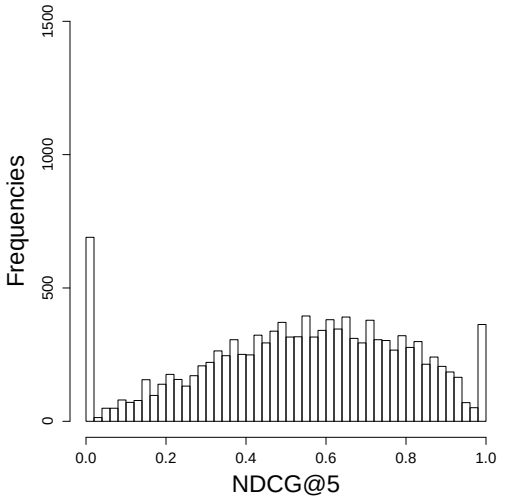
(a) DCG Errors



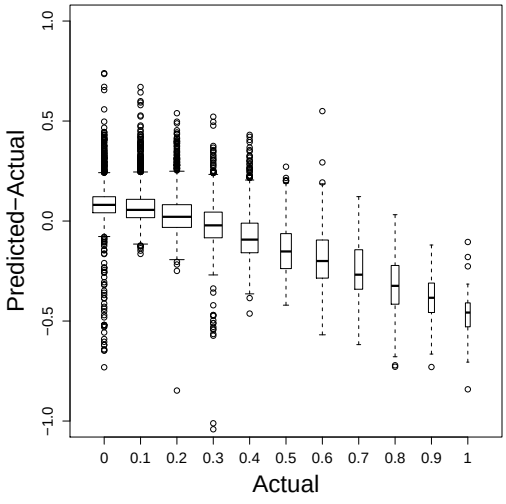
(b) NDCG Errors



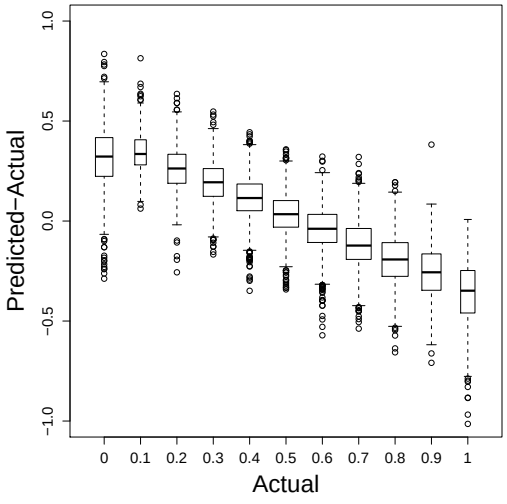
(c) DCG Distribution



(d) NDCG Distribution



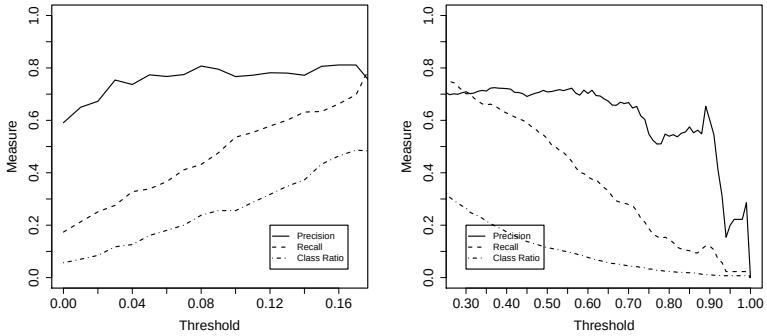
(e) DCG versus Errors



(f) NDCG versus Errors

Figure 3.3: Distribution of DCG and NDCG prediction errors.

whereas, reliable identification of *hard* queries is more useful for enabling query reformulations. In order to highlight prediction effectiveness for different types of queries, we formulate two tasks: 1) Identifying hard queries - queries whose target metric is below a specified threshold, and 2) Identifying easy queries - queries whose target metric is above a specified threshold. Using standard precision and recall metrics, we can then focus on the type of queries that are most useful for the application of interest.

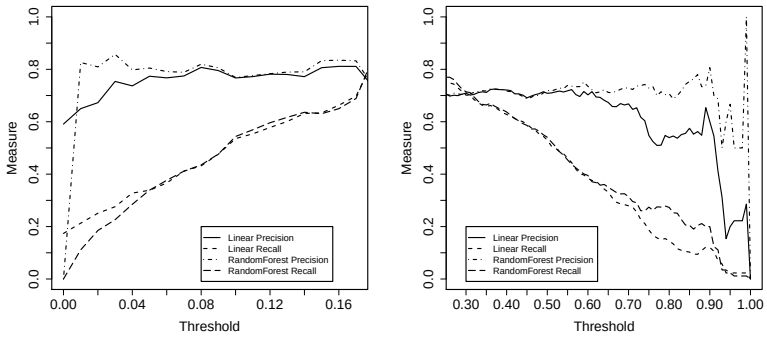


(a) DCG: Hard Queries (b) DCG: Easy Queries

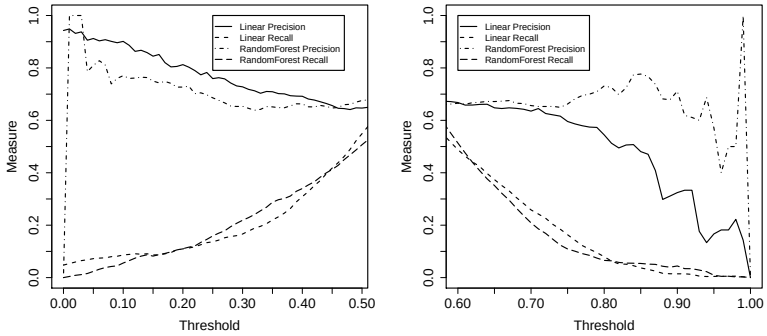
Figure 3.4: Prediction accuracy at different effectiveness regions: Precision, Recall and Class Ratio for identifying (a) Hard queries - queries whose actual DCGs are less than a specified threshold (x - axis), and (b) Easy queries - queries whose actual DCGs are greater than a specified threshold (x - axis).

Figure 3.4(a) shows precision and recall values for the task of identifying hard queries for varying thresholds. Hard queries are identified with high precision but with low recall. For example, for the task of predicting queries with DCG$\leq$ 0.08, the precision is nearly 80% but the recall is less than 50%. Further, as the threshold increases, the task becomes progressively easier, and consequently, both precision and recall improve. For finding *easy* queries, precision is much lower for finding the most *easy queries*. Note that in Figure 3.4(b) as threshold increases, the task becomes progressively harder. Nonetheless, precision drops more dramatically i.e., it is much harder to identify easy queries reliably.

Linear vs. Non-Linear Regression. Figures 3.5(a)-3.5(d) compare the effectiveness of linear and non-linear regressions. For both DCG and NDCG prediction,



(a) DCG: Hard Queries (b) DCG: Easy Queries



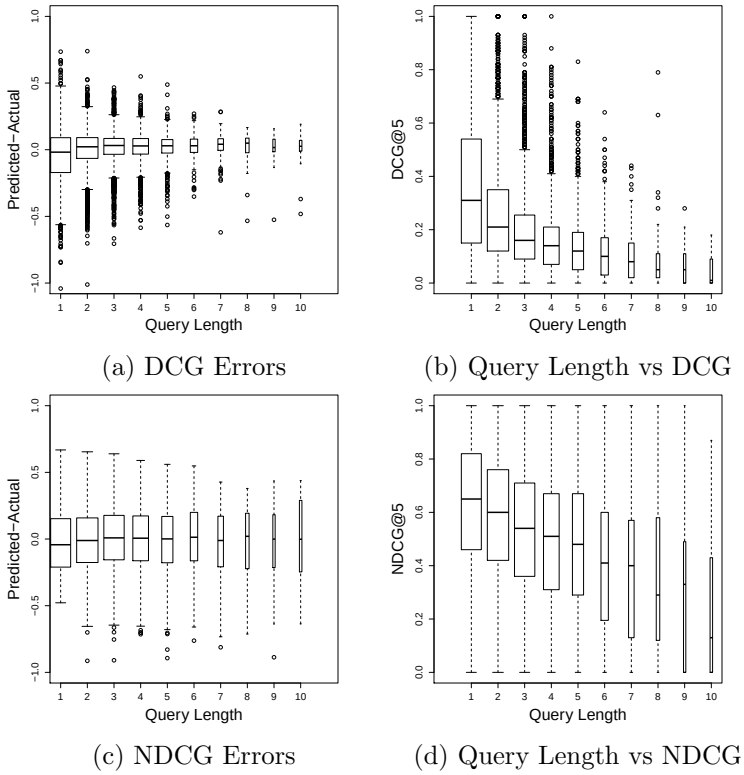
(c) NDCG: Hard Queries (d) NDCG: Easy Queries

Figure 3.5: Linear versus Non-linear prediction.

using a non-linear regressor improves prediction accuracies. Specifically, prediction for easy query prediction improves dramatically, with a relatively small loss in precision for hard query prediction. Some regression features are not monotonically related to the target metrics. In particular, NDCG features that are positively correlated for hard queries (with NDCG < 0.3) are actually negatively correlated for the easy queries (with NDCG > 0.7). We hypothesize that the random forest regression is better able to handle this non-linearity in feature correlation and hence the improved performance for easy queries.

Query Length. Predicting performance for long queries is easier than for shorter queries. Figure 3.6(a) shows a box plot of the distribution of DCG prediction errors for queries of different lengths (number of words).

As query length increases the range of DCG prediction errors decreases. This is in part because query length is an important feature in our regression. In fact it is one



(a) DCG Errors (b) Query Length vs DCG

(c) NDCG Errors (d) Query Length vs NDCG

Figure 3.6: DCG and NDCG prediction errors distribution for queries of different lengths.

of the top ten most important features in the regression. Prior works have shown that query length has a strong negative correlation with retrieval effectiveness, i.e., retrieval is less effective for longer queries compared to shorter keyword queries (Kumaran & Allan, 2007; Bendersky & Croft, 2009). Figure 3.6(b) also confirms this observation. The number of queries decrease for increasing query lengths but we note that more than 10% of the queries (i.e., more than 1200 queries) have length 5 or more which suggests that the trend of reduced prediction errors for longer queries is not entirely due to the reduction in sample size.

For NDCG, Figure 3.6(c) shows that prediction errors do not decrease for longer queries. This is because as shown in Figure 3.6(d) the range of NDCG values for long queries is much larger than the range of DCG values, even though the average NDCG values decrease as query length increases.

In the next section, we describe how these features can be used for query-dependent selection.

3.3 Learning Formulations

Given a query, a set of retrieval alternatives, and a set of features that are indicative of effectiveness of the alternatives, the goal of query-dependent selection is to select an alternative that is likely to yield the best effectiveness. Here we describe our general approach to modeling the problem of selecting alternatives, and outline three specific learning formulations that we investigate.

Problem Definition. Let $\mathbb{Q}$ be the set of training queries. For each query $Q \in \mathbb{Q}$, let $\mathbf{A}_Q = \{A_0, A_1, \cdots, A_{m-1}\}$ be the set of m alternatives of a single type. A_i corresponds to the feature vector for the alternative i, which is a concatenation of the features listed in Table 3.1. For example when choosing between retrieval models, the set of alternatives is the set of feature vectors constructed from the results retrieved using each retrieval model.

Let $T(A_i, Q)$ denote a target measure of the effectiveness of the ranking produced by alternative A_i for the query Q. The selection problem is to find an alternative $A^* \in \mathbf{A}_Q$ that achieves the highest value for the target measure as shown in Equation 5.1.1.

$$A^* = \arg\max_{A \in \mathbf{A}_Q} T(A, Q) \tag{3.1}$$

Obviously, the target measures cannot be completely specified for inferences over all possible queries, and hence we need to estimate $T(A, Q)$.

We use the features we described in Section 3.2 to provide us with estimates of effectiveness and use these estimates in three formulations for selecting between the alternatives: 1) *Independent* estimation, 2) *Difference* estimation, and 3) *Ranking*. The formulations mainly differ in their target learning functions.

Independent Estimation

Given the set of alternatives $\mathbf{A}_Q$, we estimate the performance of each alternative independently. Then, we simply select the alternative that has the highest predicted performance.

Formally, given a set of functions $h : \mathbf{A}_Q \rightarrow \mathbb{R}$, we learn a non-linear regressor h^* that minimizes the mean squared error as given by:

$$h^* = \arg\min_h \sqrt{\sum_{\forall Q \in \mathbb{Q}} \sum_{A \in \mathbf{A}_Q} \left(h(A) - T(A, Q)\right)^2}$$

Then, for a given test query Q_t, we select the alternative A^* with the largest predicted performance, i.e.:

$$A^* = \arg\max_{A \in \mathbf{A}_{Q_t}} h^*(A) \tag{3.2}$$

Difference Estimation

In the Difference formulation, we predict the difference in performance between each alternative and a designated baseline alternative, A_0. Then, we select the alternative that has the highest positive difference. If there is no alternative with a predicted positive difference, then we choose the baseline A_0.

Let $D(A_0, A, Q) = T(A, Q) - T(A_0, Q)$, denote the target measure difference between an alternative A and the baseline A_0. Given a set of functions $h_d : \mathbf{A}_Q \times \mathbf{A}_Q \rightarrow \mathbb{R}$, we learn a least-squared-errors regressor h_d^* given by:

$$h_d^* = \arg\min_{h_d} \sqrt{\sum_{Q \in \mathbb{Q}} \sum_{A \in \mathbf{A}_Q} \left(h_d\left(A_0, A\right) - D\left(A_0, A, Q\right)\right)^2}$$

Then, for a given test query, Q_t, we choose an alternative, A^* as:

$$A^* = \arg\max_{A \in \mathbf{A}_{Q_t}} h^*(A_0, A) \tag{3.3}$$

Ranking

In this formulation, the goal is to rank the alternatives in order to select the top ranking alternative. The ranking model is learned by training on pairwise preferences between the alternatives.

For each alternative $A \in \mathbf{A}_Q$, A_i is preferred over A_j if $T(A_i, Q) \geq T(A_j, Q)$. The pairwise preferences induce a partial ordering and the query at the top of the ordering is selected. This formulation fully encodes dependencies between the alternatives. Kumaran & Carvalho (2009) use this learning to rank approach to select *only* amongst reduced versions of the query on TREC collections. We formalize this approach to rank retrieval alternatives.

Let Φ denote the error function for incorrect pairwise ordering defined as follows:

$$\Phi_h(A_0, A) = \begin{cases} 1 & \text{if sign}\left(h(A) - h(A_0)\right) \neq \text{sign}\left(T(A, Q) - T(A_0, Q)\right) \\ 0 & \text{otherwise} \end{cases}$$

We want to learn a function h_r^* from the set of ranking functions $h_r : \mathbf{A} \rightarrow \mathbb{R}$ such that it minimizes the overall pairwise ordering errors, i.e.,:

$$h_r^* = \arg\min_{h_r} \sum_{Q \in \mathbb{Q}} \sum_{A \in \mathbf{A}_Q} \Phi_{h_r}(A_0, A)$$

Then, for a given test query, Q_t, we choose an alternative A^* as:

$$A^* = \arg\max_{A \in \mathbf{A}_{Q_t}} h_r^*(A) \tag{3.4}$$

Difference and *Ranking* learning formulations allow incorporation of features that characterize the relationship between the alternatives. For example, when selecting between reduced versions for long queries, we can include the estimated effectiveness of the original query as a feature. This allows us to learn from small differences in feature

values when the original query's effectiveness is low, and prefer larger differences in feature values, otherwise. As we will show later in Chapter 4, experimental evidence suggest that modeling this relationship results in improved selection performance.

3.4 Summary

In this chapter we developed a relative effectiveness estimation technique, ReEff. We described the features we use and the learning formulations we invetigate. Our experiments on a large scale web collection show that the result set-based features in ReEff are strong indicators of the effectiveness between result sets. However, ReEff is not as effective when learning from small scale collections using a small number of retrieval features.

Our analysis shows that despite high correlation of the predicted and actual effectiveness values, the prediction at the extremes is harder. In particular, prediction for harder queries is more reliable than for easy queries. Moreover, the non-linear regression using Random Forests is better than the linear regression, especially for easy queries. Based on these results, we choose the non-linear Random Forest based regressors for selection experiments. Finally, we note that prediction for longer query lengths is more reliable than for shorter queries, which suggests that query length can be a useful feature for selecting query representations.

In the subsequent chapters, we present some applications of ReEff for selecting between query representations, and retrieval models.

CHAPTER 4

QUERY REDUCTION

Query reduction is a technique for removing extraneous terms from long queries.The queries that are obtained by removing one or more terms from the original long query are called *reduced versions* or *reduced queries*. Automatic query reduction is the task of selecting between reduced versions of long queries. In this setting, we consider the original query, and the reduced versions as alternatives. Since there can be an exponential number of reduced versions, we propose a simple approximation to the selection problem. We evaluate the application of ReEff to a large collection of long queries that were issued to Microsoft Bing[1], a major web search engine. Our experiments show that for more than 25% of the cases, query reduction achieves more than 4% relative improvement over the original long queries.

In this chapter, we first describe applying ReEff to the problem of query reduction on the Web. We highlight the challenges on the web and show the potential for the simple approximation to query reduction. Then, we describe the additional query based features we include in ReEff present experimental evidence for the performance of ReEff on a large collection of web search queries. We then present two extensions to ReEff, thresholding and results interleaving as two mechanisms for reducing the risk involved in selection. Finally, we present an analysis of the selection results to highlight the nature of improvements that ReEff provides.

[1]http://www.bing.com

4.1 Approach

The effectiveness of retrieval systems is typically lower for longer queries than for shorter keyword queries (Kumaran & Allan, 2007; Bendersky & Croft, 2009). This is in part because long queries contain extraneous terms, which when down-weighted or removed lead to improved retrieval effectiveness. For example, the query *easter egg hunts in northeast columbus parks and recreation centers* performs moderately well on three most popular web search engines (Microsoft Bing, Yahoo!, and Google). If we remove (or down-weight) the terms *and recreation centers*, we observe a perceptible improvement in the quality of results in all three search engines[2].

The idea of automatically identifying and removing extraneous terms from long queries is referred to as *query reduction*. This approach showed great potential for improving performance on long queries on TREC collections (Kumaran & Allan, 2007; Kumaran & Carvalho, 2009). In this work, we focus on automatic query reduction for long queries on the web.

Automatic query reduction on the web involves two main challenges. Existing content based measures do not work as well on the web (Chapter 3) and they are not as efficient. Efficiency is particularly important for web search engines, where millisecond level latencies are expected. We show that ReEff can effectively address these challenges by using effective and efficient features for selecting reduced versions. Below, we first formalize the query reduction problem and describe the application of ReEff to this problem.

4.1.1 Problem Definition

The retrieval alternatives in this case are the original query and its reduced queries. Given a query $Q = \{q_1, \cdots, q_n\}$, the set of alternatives $\mathbf{A}_Q$ is the set containing all non-empty subsets of terms from query (including the original query Q), which

[2]Observed as of January 2010.

is the power set $\mathcal{P}^Q$. Let $T(P, Q)$ denote the effectiveness of the alternative (the representation) P for query Q. Then, the query reduction problem is to find an alternative $P^* \in \mathcal{P}^Q$ that achieves the highest value for the target measure, as shown in Equation 4.1 .

$$P^* = \arg\max_{P \in \mathcal{P}^Q} T(P, Q) \tag{4.1}$$

Note that this problem statement allows the original query Q to be selected as well.

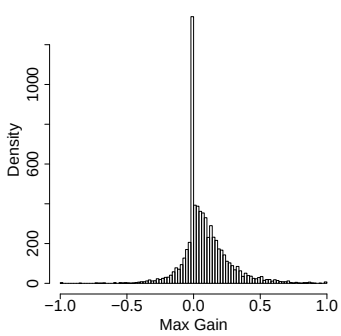
Efficiency is a key challenge for query reduction. Because of the large number of possible reduced queries in $\mathcal{P}^Q$, enumerating and evaluating all possibilities is not feasible. We use a simple approximation of the query reduction problem. Instead of considering all possible reduced versions, we only consider those that differ from the original query Q by only one term. That is, instead of using the entire power-set $\mathcal{P}^Q$, we use a restricted version $\mathcal{P}_1^Q = \{P | P \in \mathcal{P}^Q \wedge |P| \geq |Q| - 1\}$. Thus, if the original query had n query words, we only need to consider the n reduced versions and original query Q.

4.1.2 Potential

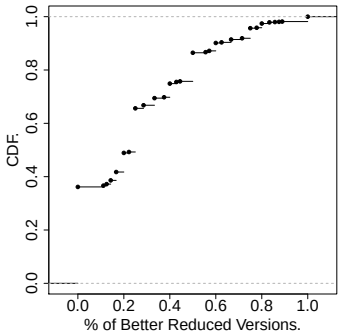
Despite the obvious limitation of ignoring a large number of potentially better reduced versions, this simple approach can yield dramatic performance improvements. On a large collection of more than 6400 Web queries, we find that an oracle that chooses between an original long query and its reduced by-one versions can achieve more than 10% absolute gain in NDCG@5.

In order to achieve this gain, we need to reliably identify reduced versions whose performance is better than the original query. We analyze the distribution of gains shown in Figure 4.1 to illustrate the potential impact of query reduction.

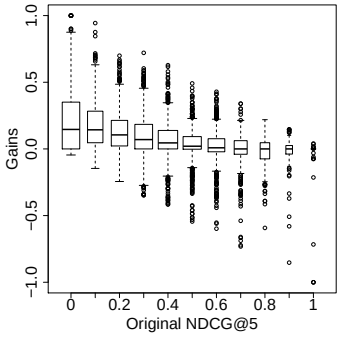
Figure 4.1(a) shows the gains (or lack thereof) when using an oracle that always selects the best reduced version. For most queries it is possible to obtain some positive improvements. However, the problem of selecting the best reduced version or



(a) Max Gains: The maximum gain possible using reduced queries alone.



(b) Distribution of reduced queries that are better than the original query.



(c) Original versus Gains: Distribution of gains for original queries of different effectiveness levels.

Figure 4.1: Query reduction potential: Distribution of NDCG@5 gains when choosing between reduced versions that differ from the original query by one term.

even some reduced version that is better than the original query is non-trivial. Figure 4.1(b) illustrates this point. More than 35% of the queries have no reduced queries that are better than the original query and for 80% of the queries fewer than half of the reduced versions are better than the original. These observations suggest that selection can be a risky process. We consider thresholding and interleaving results as two simple approaches to mitigate the risk involved in selecting reduced versions.

Finally, as shown in Figure 4.1(c), if the original query has poor performance, then it is more likely for some reduced version to be better than the original query.

Conversely we are not likely to find reduced versions of well-performing queries that provide substantial performance gains. We utilize this relationship in our learning formulation to obtain further improvements in selection performance.

4.1.3 Features

We use two types of features to perform query reduction, which we refer to as *result set-based* features and *query-based* features. Table 4.1 lists the set of features used.

Result set-based features - The result set-based features are aggregates of ranker scores and retrieval scores. Chapter 3 describes the retrieval-based features in detail. Here, we describe the additional query-based features.

Query-based features - The query-based features are simple lexical and category indicators computed over the queries themselves. Different types of queries have different levels of effectiveness. For example, retrieval is often more effective for navigational and transactional queries than for informational queries. Also, different types of queries may benefit from different alternatives. For example, dropping terms from a navigational query can be detrimental. On the other hand, long informational queries often benefit from dropping terms. Therefore, information about the type of query can be useful indicators of retrieval effectiveness, as well as for determining when reduction can be useful.

Instead of attempting to categorize queries into these different types, we directly use features that are indicative of these categories. For example, presence of a URL is a strong indicator of navigational queries. Similarly, we use other lexical features such as stop words, dates, numbers, as well as location features that denote the presence of town, city, or state names as part of the feature set. Additionally, we use query length as a feature. Query length has a strong negative correlation with performance. Our experiments on a large web collection and prior work on long web

Table 4.1: Features used for query reduction. *Pos* (positional features) indicates that the values of the corresponding feature at each rank (position) is used as an independent feature. *Agg* (aggregate features) indicates that statistical aggregates (mean, max, standard deviation, variance and coefficient of dispersion) of the values for the top-k documents are also used as independent features. Additionally, we perform a min-max normalization to rescale all features between $[0, 1]$.

Type	Feature Name	Description	Variants
Query-based			
	URL	Binary: Does query contain an URL?	-
	Stop-words	# of stop-words in query	-
	Number	Does query contain a number?	-
	Location	Does query contain a town, city name or a state name ?	-
	Query Length	# of words in query	-
Result set-based			
	LR	LambdaRank score of top-k documents	Pos. and Agg.
	BM25	Okapi-BM25 score of top-k documents	Pos. and Agg.
	Click-based	Click-through counts and other variants	Agg.
	Static Scores	Page-rank like scores of the top-k documents	Agg.

queries show that retrieval effectiveness is lower for longer queries than for shorter keyword queries (Bendersky & Croft, 2009).

4.1.4 Formulations

We use *Independent*, *Difference*, and *Ranking*, the three formulations that are described in Chapter 3. For *Independent* and *Difference* we use Random Forests to build regression models. The *Independent* regression model is trained to predict the effectiveness of a given query representation. The representation with the highest predicted effectiveness is selected. The *Difference* regression model predicts the difference in effectiveness between each reduced query and its original query. The reduced query with the largest predicted positive difference is selected. If no such re-

duced query exists, then we choose the original query. The *Ranking* model is trained to predict whether a reduced version is better than its original query. The top-ranked reduced version when sorted in descending order of the predicted values is used if its score is positive. Otherwise, the original query is used.

4.2 Experiments

We conduct query selection experiments is to demonstrate the utility of ReEff for automatically selecting a representation for the original query. ReEff can choose to use the original query or one of the reduced versions obtained by dropping one term at a time.

We use a large proprietary collection of 6461 long Web queries that were issued to the Microsoft Bing search engine. First, we extract queries that were of length four or more from the Bing search engine query logs. Then, we obtain a frequency weighted random sample of 6461 queries to select a representative sample from the logs.

For all queries, both original and reduced, we obtain results using LambdaRank (Burges et al., 2007), an effective learning to rank algorithm. The learning to rank algorithm combines several types of features, including BM25 (Robertson et al., 1994) based features, click-through based features, and static features like page rank (See Appendix for a more detailed listing of these features). The top five results for each representation are pooled and judged for relevance. The resulting collection of queries and judged documents are used as the data set for the selection experiments.

We evaluate the performance of ReEff using NDCG@5 (Järvelin & Kekäläinen, 2002).

For *Independent* and *Difference* formulation, the goal of learning is to find real-valued functions that predict the target metric, and the differences in the target metric, respectively. For both formulations, we use non-linear regression with the

Random Forests (Liaw & Wiener, 2002) algorithm[3]. We used the *randomForest* package available from R with default parameters for the number of trees (500) and the number of features to select at each iteration (50). For the *Ranking* formulation, we use RankSVM (Joachims, 2002) to train on pairwise preferences between the original query and the reduced query. We use the SVMLight implementation[4] with a linear kernel and default values for the cost parameter C[5]. For all three problem formulations we use five-fold cross validation procedure for training and evaluating the learning models.

4.3 Results

Table 4.2: Effectiveness of ReEff for query selection. The baseline, using the original query always, has an overall NDCG@5 of 38.19. Bold face indicates the highest value, and * indicates statistically significant improvement over the baseline ($p < 0.05$ on a two-tailed paired t-test).

	Overall NDCG@5	**Affected Queries**	**Improved Queries**	**Hurt Queries**	**Subset NDCG Gain**
Original	38.19	NA	NA	NA	NA
Independent	35.18	4567 (70%)	1583	2346	- 4.26 (-12%)
Difference	**38.63***	1761 (27%)	513	427	+ 1.61 (+4.2%)
Ranking	38.50*	612 (9%)	245	212	+ 4.64(+12.1%)

Table 4.2 shows the selection performance of the different problem formulations. We describe the selection performance in terms of the overall average NDCG@5 and the gain in the subset of queries for which the formulations choose a reduced query, shown by the column *Subset NDCG gain*.

[3]We experimented with a linear regression model with and without L1 constraints for effectiveness estimation and found the non-linear Random Forest regression to perform the best.

[4]`http://svmlight.joachims.org/`

[5]C is the trade-off between the error and the margin, set to inverse square root of number of instances.

Difference and *Ranking* yield small but significant average improvements over using the original long queries alone. *Difference* and *Ranking* affect a smaller proportion of the queries but their selections lead to improved average performance compared to using the original query alone.

On the other hand, *Independent* performs worse compared to the original queries. The poor performance of *Independent* is partly due to its aggressive selection of reduced versions. *Independent* selects reduced versions for a large proportion of queries (nearly 70%) but its selections lead to worse performance in in far more queries.

Independent also does a poor job of ranking the reduced queries. Table 4.3 shows the average performance if we always selected the top-ranked reduced query according to each formulation. The top ranked reduced queries selected by *Difference* and *Ranking* are substantially better than those selected by *Independent*. This is in part because *Independent* only indirectly models the ranking of reduced queries. The regression training seeks to minimize the mean-squared errors of the predicted and actual NDCG@5 values for each query independently. *Difference* and *Ranking* formulations on the other hand, directly predict whether each reduced version is better than the original query. In particular, the regression for *Difference* seeks to directly predict the difference in effectiveness between the reduced query and the original query.

Table 4.3: Average NDCG@5 of the top-ranked reduced queries according to each formulation.

Orig	***Independent***	***Difference***	***Ranking***
38.19	34.50	37.22	36.17

Overall, the average performance of selection is better than that of using the original queries alone, but selection is a risky process which helps some queries and hurts others. We explore two methods to reduce the risk involved in selection.

4.3.1 Thresholding

We conduct thresholding experiments to investigate the effect of controlling the number of queries for which reduced versions are selected. We use a selected reduced query only if the difference between its predicted effectiveness and the original query's predicted effectiveness exceeds a specified threshold. Since the *Difference* and *Ranking* formulations already predict differences, we directly apply the thresholds to the predicted differences i.e., we use the selected query only if its predicted difference exceeds the specified threshold. We use the training data to learn thresholds that maximize the overall NDCG@5.

Table 4.4: Effectiveness of thresholding for query selection. The baseline, using the original query always, has an overall NDCG@5 of 38.19. Bold face indicates the highest value, and * indicates statistically significant improvement over the baseline ($p < 0.05$ on a two-tailed paired t-test).

	Overall NDCG@5	**Learned Threshold**	**Affected Queries**	**Improved Queries**	**Hurt Queries**	**Subset NDCG Gain**
Independent	**38.64***	0.2	457 (7%)	209	149	+ 6.33 (+16.5%)
Difference	38.63*	0.0	1761 (27%)	513	427	+ 1.61 (+4.2%)
Ranking	38.50*	0.0	612 (9%)	245	212	+ 4.64(+12.1%)

Table 4.4 shows the results of thresholding for the three formulations. For each formulation, we present results at the threshold that was learned during training.

Only *Independent* benefits from thresholding in terms of average performance. *Independent* achieves the best average performance at a threshold of 0.2, i.e., when the predicted difference between the selected reduced query and the original query is greater than 0.2. On the other hand, *Difference* and *Ranking* learn a threshold of 0.0, which is the same as not performing any thresholding at all. As a result, *Difference* and *Ranking* do not achieve any additional improvements.

Independent benefits from thresholding as it selects fewer reduced versions which are more likely to yield improvements. *Independent*'s average performance improves substantially and its overall performance is comparable to *Difference*. This also illustrates the benefit of targeting the relative differences between alternatives. Despite

the similar average performance over the entire set of queries, the three formulations provide different types of improvements.

First, *Independent* and *Ranking* select reduced versions for fewer queries, whereas *Difference* selects reduced versions for more queries. Second, *Independent* and *Ranking* achieve higher subset gains compared to *Difference*but the fraction of queries for which they produce a negative impact is also substantially higher. For nearly 41% of the queries, *Independent* selects reduced queries that are worse than using the original query. *Difference* on the other hand selects reduced queries that are worse than original only in 25% of the queries. This suggests that directly learning to predict the differences is more robust than a two-staged approach of independent estimation followed by threshold learning on the differences. We further analyze this trade-off between percentage of queries impacted and the quality of impact in Section 4.4.

4.3.2 Results Interleaving

In addition to thresholding, we also conduct interleaving experiments. In all three formulations, if a reduced version is selected for the given threshold, we interleave its results with the results of the original query. Furthermore, we decide the order of interleaving based on the predicted performance. If the original query's predicted performance was higher than that of the reduced version then interleaving begins with the original query. Otherwise we begin interleaving with the reduced query. Because interleaving combines results from the original query and the top-ranked reduced version, it can yield gains even in cases where the top-ranked reduced version's predicted performance is lower than that of the original query.

Table 4.5 shows the gains achieved by the interleaving results. *Difference* achieves the best overall gains, whereas *Independent* achieves the best subset gains. *Difference* and *Ranking* are more aggressive than *Independent* in selecting reduced versions for interleaving. In fact, the thresholds that maximize the average performance of

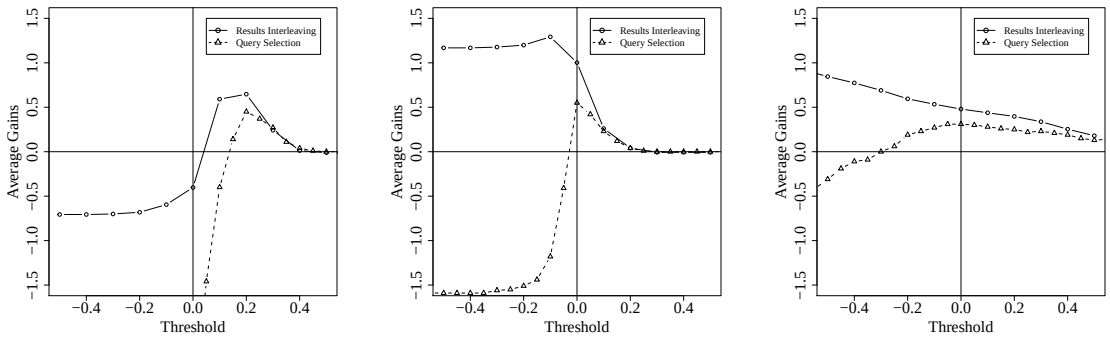
Table 4.5: Results Interleaving at the best thresholds for the three formulations. For the NDCG Gain row, bold face indicates the highest value, and * indicates statistically significant improvement over original NDCG@5 ($p < 0.05$ on a paired t-test).

	Overall NDCG@5	**Learned Threshold**	**Affected Queries**	**Improved Queries**	**Hurt Queries**	**Subset NDCG Gain**
Independent	38.89*	0.2	457	228 (4%)	139 (2%)	+ 9.89
Difference	**39.49***	-0.2	6435 (31%)	1979(31%)	1949(25%)	+ 1.61
Ranking	39.16*	-0.8	5258	1620 (25%)	1612(25%)	+1.19

Difference and *Ranking* are negative i.e., they select reduced versions for interleaving even when there is a higher risk of failure. This is because they achieve better ranking of reduced versions compared to *Independent* (see Table 4.3), thus allowing more queries to benefit from interleaving. As a consequence, *Difference* and *Ranking* both have a positive impact on a large number of queries, 31% and 25% respectively, whereas *Independent* provides positive gains for only 4%.

In terms of the average performance, interleaving is better than selection at most thresholds. Figures 4.2(a), (b), and (c), show the performance of query selection and query selection at different thresholds. For all formulations interleaving results is better than query selection at most thresholds. Interleaving has two advantages. First, in the case of erroneous choices, interleaving ensures that at least some of the original query's results are mixed in with the chosen reduced query's results, which reduces the risk of hurting performance. Second, when good reduced queries are chosen, interleaving can also benefit from the fusion of different relevant results from the original and the reduced queries.

Interleaving and selection yield different types of benefits. Figure 4.3 shows the cumulative density function (CDF) of the differences in NDCG@5 of query selection (shown by the thicker line) and Results Interleaving (shown by the thinner line) with respect to the baseline (using the original query always). It shows that interleaving yields large positive gains for fewer queries compared to query selection. However,



(a) Independent: NDCG@5 (b) Difference: NDCG@5 (c) Ranking: NDCG@5

Figure 4.2: Interleaving Results.

Results Interleaving also results in fewer losses overall, which also explains the overall average improvements when using results interleaving.

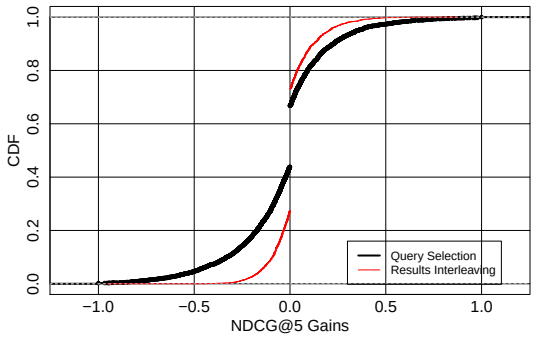


Figure 4.3: Selection versus interleave gains for *Independent*. The *Choice* curve shows the CDF of gains when using query selection and the *Interleave* curve shows the CDF for gains using results interleaving.

4.3.3 Long Gov2 Results

We also conduct small-scale evaluations on the publicly available TREC Gov2 Web collection (Long Gov2) using BM25 as the ranking function. We conduct experiments on the TREC Gov2 Web collection using the description portion of the topics as the queries. Table 4.6 shows the results for query selection on a small collection of 149

queries on the TREC Gov2 Web collection. Unlike the Web collection, we only use the aggregates of BM25 scores as features for the query reduction experiments.

Difference achieves better performance compared to *Independent*. As with the larger web collection, we find that *Independent*'s selection of reduced queries is ineffective and hurts more than it helps and applying thresholding improves its performance substantially. *Ranking* choses reduced queries only for six queries, as a result of which its overall impact is limited. Due to limited overall improvements on a small sample size, none of the observed differences are statistically significant. However, the overall performance trends confirm the observations on the larger web collection, despite the differences in the type of queries, the limited amount of training data and features used.

Table 4.6: Long Gov2 Query Reduction Results.

	Overall NDCG@5	Affected Queries	Improved Queries	Hurt Queries	Subset NDCG Gain
Original	34.44				
Independent	31.64	109	35	44	-3.83
Difference	35.30	129	48	40	0.99
Ranking	34.71	6	2	1	6.7
Thresholding					
Independent	34.55	16	6	6	1.02
Difference	35.30	129	48	40	0.99
Ranking	35.99	127	35	28	1.81

4.4 Analysis

We further analyze the results to better understand the contributions of different types of features, distribution of gains and the nature of the improvements obtained through ReEff.

4.4.1 Feature Importance

We conduct two selection experiments to demonstrate a) the utility of the query-based features that we added to ReEff and b) the utility of the original query's effectiveness for *Difference*.

Table 4.7 shows the results of these experiments. Using the result set-based features alone yields provides most gains over the original queries. Adding the query-based features provides small additional improvements.

As we discussed earlier in Section 4.1, we are unlikely to find improvements for original queries that are already highly effective. To model this observation, we simply use the predicted NDCG@5 of the original query as a feature in the *Difference* formulation[6]. As shown by the *Estim+* column, this leads to further improvements in average performance. This illustrates the potential of the *Difference* formulation to encode relationships between alternatives.

Table 4.7: Feature importance for *Difference*: *Orig* — using original query, with no query replacement. *Result set-based* — Using result set-based features alone. *+Query-based* — adding query-based features to the result set-based features. +Estim — adding estimate of the performance of the original query to *+Query*.

	Orig	**Result set-based**	**+ Query-based**	**+ Estim**
NDCG@5	38.19	38.52	38.63	38.73
Subset Gain	0	1.30	1.61	2.05

4.4.2 Distribution of Selection Gains

Query reduction results in dramatic gains on some subset of queries but also incurs losses on some queries. Here we investigate how the gains (and losses) are distributed.

Potential versus Achieved Gains. All three formulations provide improvements when there is a large potential for improvement. Figure 4.4 shows the distribution of the gains achieved by *Independent* in relation to the best gains that an

[6]We take care to ensure that the effectiveness estimates for training are learned only using the training folds and not the test folds.

oracle can achieve. In most cases when the potential for improvement is large, *Independent* formulation achieves large improvements. Also, when the potential is greater than 0.8, *Independent* always results in some positive improvement. The large gains achieved by the formulations are in part due to two factors.

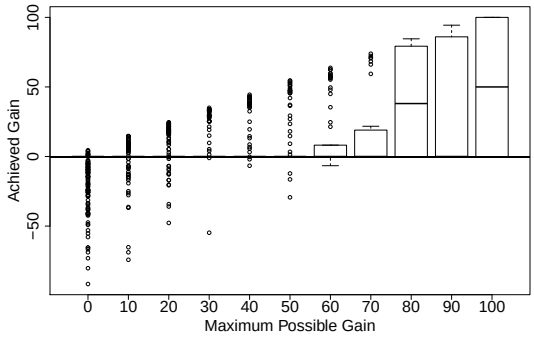


Figure 4.4: Oracle versus Achieved Gains: Boxplot showing the distribution of gains achieved by query selection using *Independent* in relation to the best possible gains that can be achieved by an oracle.

First, the formulations are able to detect large differences in NDCG@5 more reliably. The distribution of absolute prediction errors for the regression used by *Independent* is shown in Figure 4.5. The histogram shows the frequencies of absolute difference between the predicted and the acutal values of NDCG@5 for all queries (including reduced versions). Most prediction errors are smaller, and very few errors are larger than 50 points (less than 3%). This suggests that smaller differences in NDCG@5 are harder to capture given the range of prediction errors, while larger differences can be captured more reliably.

Second, for queries with large gains, the problem of choosing a reduced version becomes easier. Figure 4.6 shows the distribution of the number of reduced versions that are worse than the original query. When there are large gains there are fewer reduced queries that are worse than the original, which makes the problem of finding better reduced versions easier.

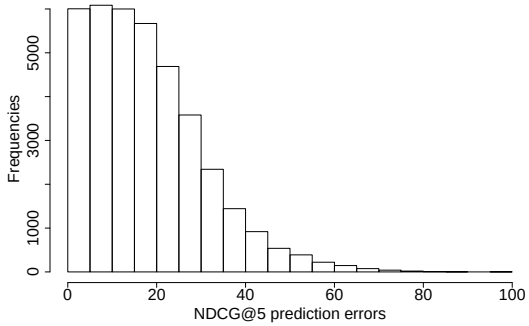


Figure 4.5: NDCG@5 Prediction Errors: Frequency histogram of difference between predicted and actual NDCG@5 values.

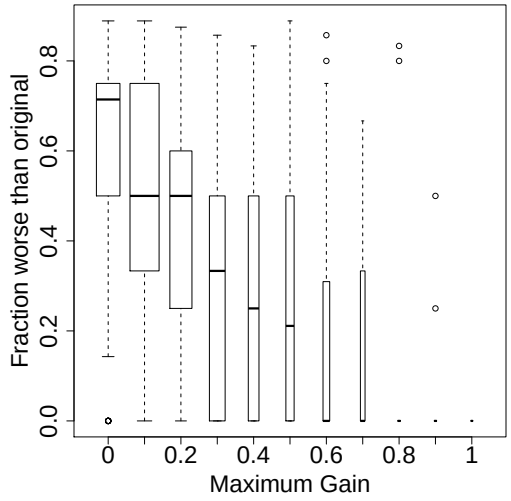
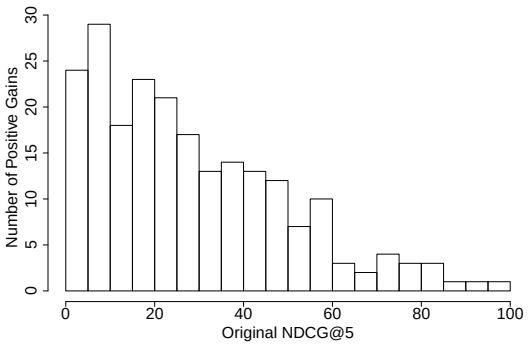
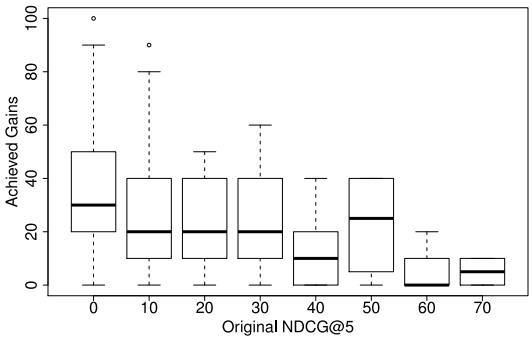


Figure 4.6: Boxplot showing distribution of reduced versions that are worse than the original query (y-axis) against the maximum possible gains (x-axis).

Improving Poorly Performing Queries. Most gains are achieved for poorly performing original queries. Figure 4.7(a) shows the histograms for the number of queries that achieve positive gains against the effectiveness of the original query. Clearly, most of the gains are achieved for queries whose original NDCG@5 low. Nearly 75% of the queries that benefit from *Independent* are queries with NDCG@5 $\leq$ 40. Further, the magnitude of the gains for the poorly performing queries are higher than for well performing queries as shown in Figure 4.7(b). Thus, unlike



(a) Number of queries at each effectiveness level which achieved positive gains.



(b) Boxplot showing magnitude of achieved gains for original queries of different effectiveness levels.

Figure 4.7: Performance of *Independent* for original queries of different effectiveness levels

traditional techniques such as pseudo-relevance feedback (**?**), query reduction delivers improvements where it matters most.

This is in part because poorly performing original queries often have large potential for improvements (as shown in Figure 4.1) and as we showed earlier, this usually results in some improvements through selection. The improvements for poor queries is also in part due to better effectiveness estimation. As we discussed in Chapter 3, the effectiveness estimations are more reliable for poorly performing queries than for well performing queries.

4.4.3 Quality of Impact

To better understand the relationship between the number of queries affected versus the average improvement in performance, we explore the behavior of the different formulations at various thresholds. In general, we expect increasing thresholds to

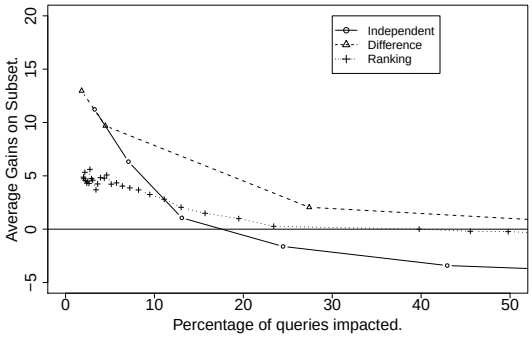


Figure 4.8: Number of queries affected versus improvements in subset gains.

cause fewer reduced versions to be chosen (lower recall), but to also increase the likelihood of improving over the original query (higher precision). Figure 4.8 shows this precision-recall trade-off in terms of the gains achieved on the subset of affected queries against the percentage of queries affected. As expected for all three formulations, the average performance on the subset is substantially high for a small percentage of queries, and performance decreases as more queries are affected. *Independent* and *Difference* achieve more than 10 points absolute improvement over the original queries on a subset of 5% of the queries. Compared to *Independent*, *Ranking* achieves smaller subset gains but its positive performance is retained over a large fraction of queries. For covering large percentages of queries, *Difference* and *Ranking* provide better trade-off in terms of the possible subset gains, whereas *Independent* outperforms *Ranking* and is comparable to *Difference* when covering smaller percentages.

4.5 Summary

In this chapter, we show that ReEff can be effectively applied for selecting reduced representations of long web queries. Our experiments on a large collection of long web queries show *Difference* and *Ranking* formulations obtain better average performance

compared to *Independent*. The results also show that thresholding and interleaving can help mitigate the risks involved in selection.

Independent's average performance benefits from thresholding because thresholding focuses on relative differences. On the other hand, since *Difference* and *Ranking* already focus on relative differences they do not obtain any additional benefits. All formulations benefit from results interleaving. *Difference* and *Ranking* benefit more from interleaving because they select better reduced queries compared to *Independent*.

Our analysis shows that query-based features we described in this chapter and the use of original query's effectiveness estimates provide small but additive improvements to selection performance for *Difference*. In terms of the types of gains obtained, we find that query reduction works when there is large potential of improvement, and it benefits poorly performing queries the most. Query selection using ReEff also provides a trade-off mechanism between the number of queries affected and the subset gains possible.

CHAPTER 5

RANKING ALGORITHMS

In this chapter we describe the application of ReEff for query-dependent selection of ranking algorithms (also referred to as rankers) for web search. Specifically, we compare the *Difference* and *Independent* formulations to illustrate the benefits of directly modeling relative differences. We develop additional result-set based features, including average feature similarity and fraction of overlapping documents for ReEff. Our experiments on a large learning-to-rank data set show that the *Difference* formulation is effective for selecting rankers, and outperforms a state-of-the-art ranker selection technique. The results show that *Difference* is also useful for improving three fusion techniques which merge results from multiple rankers.

Many web search engines combine several features using learning to rank algorithms. Learning to rank algorithms (rankers) differ based on the features they use, the type of objective functions they target, as well as the optimization techniques they use to find parameter settings. For example, RankSVM minimizes a pairwise objective function using convex optimization (Joachims, 2002), whereas AdaRank (Xu & Li, 2007) and Co-ordinate Ascent (Metzler, 2007) use direct optimization for rank-based metrics such as MAP. Given these differences in the rankers, their relative performance can vary for different queries, even though their average performance can be similar. Ideally, a query-dependent selection of rankers will help improve performance over a fixed choice.

There are two broad approaches for leveraging multiple rankers:

- Selection - The idea behind ranker selection is to select the best ranker for each query. Several query-dependent modifications have been proposed for selecting appropriate ranking functions for each query. While some focus on identifying the subset of training data to train query-specific ranking functions from, others focus on designing loss functions that are query-specific (Bian et al., 2010). In recent work, Peng et al. (2009) use the training data to identify nearest-neighbor queries, which are then used to estimate the expected performance of each ranker.

  In contrast, we focus on a direct approach for selecting the best ranker for each query using ReEff. We conduct ranker selection experiments using ReEff on a large learning-to-rank data set and demonstrate that this approach yields substantial improvements (more than 10% relative improvement on about 5% of the queries) over using a single fixed ranker. We find that when using ranker based and retrieval based features, modeling relative effectiveness of rankers is better than modeling independent effectiveness.

- Fusion - The idea behind fusion is to combine the results from the available rankers and re-rank the combined results. In this work, we show that query-dependent selection can improve fusion in multiple ways.

  We find that fusing results from all available rankers is not always beneficial. Further, fusion when useful, yields different types of benefits compared to selection: Fusion has lower impact on a larger subset of queries, whereas selection typically has a higher impact but on a smaller subset of queries,. To leverage the different benefits of selection and fusion, we conduct ranker fusion experiments, where we use query-dependent selection of rankers to augment fusion in three different ways: First, we find that for some queries fusion is beneficial, whereas for others selecting an individual ranker is better. Therefore, selectively apply-

ing fusion can improve over applying fusion for all queries. Second, we show that selecting a subset of rankers for fusion yields improvements over using all available rankers for fusion. Third, we show the weights induced by ReEff can be useful for weighted merging of the results from the rankers. Our experiments show that relative effectiveness estimation using ReEff can be effective for improving three state-of-the-art fusion techniques.

The remainder of this chapter is laid out as follows. Section 5.1 describes the application of ReEff for ranker selection including the additional features, and evaluation. Section 5.2 describes how we use ReEff to improve ranker fusion. Finally, Section 3.2.4 presents some analysis of the types of improvements that are obtained through ranker selection and fusion.

5.1 Ranker Selection

Recall that the main idea behind ReEff is to learn a regression model that can predict the difference in effectiveness between an alternate ranker and a baseline ranker. If there are multiple alternate rankers, then the alternate ranker with the highest positive predicted difference is selected. If no alternate ranker has a positive predicted difference, then we choose the baseline ranker itself. By modeling the relative difference in effectiveness, we learn to estimate the performance of the alternate rankers in relation to the baseline ranker. Thus, unlike independent estimation of effectiveness, modeling relative effectiveness allows us to capture dependencies between the performance of the rankers for each query.

5.1.1 Approach

Let $\mathbf{R} = \{B\} \cup \mathbf{A}$, be the set of m available rankers, where B is the baseline ranker and $\mathbf{A} = \{A_1, \cdots, A_{m-1}\}$ is the set of alternate rankers. the ranking produced by the ranker $R \in \mathbf{R}$ for query Q. Also let $T(R, Q)$ denote the effectiveness of the ranking

produced by the ranker $R \in \mathbf{R}$, which is measured using a target effectiveness metric such as average precision. Then, for each query Q, the ranker selection problem is to find a ranker, R^* that achieves the highest $T(R^*, Q)$.

$$R^* = \arg\max_{R \in \mathbf{R}} T(R, Q)$$

Since relevance information is not available for all queries, we need to estimate $T(R, Q)$ for unseen queries.

Independent

For the *Independent* formulation we learn a model that independently estimates the effectiveness of each ranker, $T(R, Q)$. Given a set of training queries $\mathbb{Q}$ with relevance information and a set of regression functions $h_i : \mathbf{D} \rightarrow \mathbb{R}$ that approximate $T(R, Q)$, we learn a regression function h_i^* that minimizes the squared errors between the predicted and actual effectiveness as follows:

$$h_i^* = \arg\min_{h_i} \sqrt{\sum_{Q \in \mathbb{Q}} \sum_{A_i \in \mathbf{R}} \left(h_i\left(R_i, Q\right) - T\left(R_i, Q\right)\right)^2}$$

Then, for a given test query, Q_t, we choose the ranker, R^* as:

$$R^* = \arg\max_{R \in \mathbf{R}} h_i^*(R, Q_t)$$

Difference

For the *Difference* formulation, instead of learning a regression model to independently estimate $T(R, Q)$, we learn a regression model that estimates the relative effectiveness, $T_d(R, B, Q) = T(R, Q) - T(B, Q)$. Given a set of training queries $\mathbb{Q}$ with relevance information and a set of regression functions $h_d : \mathbf{D} \times \mathbf{D} \rightarrow \mathbb{R}$ that ap-

proximate T_d, we learn a regression function h_d^* that minimizes the sum of the squared errors as follows:

$$h_d^* = \arg\min_{h_d} \sqrt{\sum_{Q \in \mathbb{Q}} \sum_{A_i \in \mathbf{A}} \left(h_d\left(A_i, B, Q\right) - T_d\left(A_i, B, Q\right)\right)^2}$$

Further, for all $q \in \mathbb{Q}$, we set $h_d^*(B, B, Q) = 0$.

Then, for a given test query, Q_t, we choose the ranker, R^* as:

$$R^* = \arg\max_{R \in \mathbf{R}} h_d^*(R, B, Q_t)$$

The maximization will select the baseline ranker B as R*, if no alternate ranker A_i has $h_d^*(A_i, B, Q_t) > 0$ for query Q_t.

5.1.2 Features

To learn the regression function for *Independent* (h_i^*), we first construct feature vectors that represent the ranking of each ranker independently. To learn the regression function for *Difference* (h_d^*), we first construct feature vectors that can represent the ranking of both the baseline ranker, B and the rankings of each alternate ranker, A_i. Then, we use the difference of these feature vectors as the final regression feature vector.

We use ReEff's result set-based features we described in Chapter 3. First, we use the scores assigned by the rankers as indicators of document relevance, which in aggregation can be effective for representing the quality of a ranking. Second, we use the features that are used by the rankers themselves to generate more aggregate features. The features used for ranking are also designed to reflect document relevance and are intimately related to the performance of each ranker. Table 5.1 lists the features that we use to represent each ranking.

Table 5.1: ReEff features: pos. refers to value of the feature each rank (e.g. the score at a given rank). hmean refers to harmonic mean and gmean refers to geometric mean. var, sd and cd refer to variance, standard deviation, and co-efficient of dispersion respectively.

Type	Feature Name	Description	Variants
Ranker Features	Alternate ranker id	Nominal feature that is identifies the type of alternate ranker.	None
	Scores of top k ranked documents	Un-normalized scores of the top k documents.	pos., min, max, mean, hmean, gmean, var, sd, cd, skew, and kurtosis
Retrieval Features	Features of top k ranked documents	Aggregates of the feature values of the top k ranked documents.	min, max, mean, hmean, gmean, var, sd, cd
	Average feature similarity	Average similarity of each document vector to the centroid of the top k document vectors	Un-normalized and L2 normalized.
	Overlapping documents.	Fraction of overlap in the top k ranked documents in the two rankings.	None

For the ranker based features, we use the ranker id itself as a nominal and the scores of each top K document as individual features. We also use aggregates such as min, max, mean, variance, standard deviation and co-efficient of dispersion. Higher scores for top K documents and higher mean aggregates can indicate highly effective rankings, whereas higher variance can indicate poor effectiveness. Further, we use two additional variants of the mean, harmonic and geometric means and two higher order descriptive statistics, skew and kurtosis, to characterize the distribution of the ranker scores. Skew measures the symmetry around the mean (or a lack thereof) of the score distribution around the mean, whereas kurtosis measures the peakedness of the distribution.

For the retrieval-based features, we use the statistical aggregates for each retrieval feature and also use two additional measures, 1) *average feature similarity*, and 2) *fraction of overlapping documents*, which capture the intra-ranking similarity and inter-ranking similarity, respectively. To measure *average feature similarity* we treat each top ranked document as a n-dimensional feature vector and construct the centroid of the top k documents. Then, we compute the average of the distances of

each top ranked document to this centroid[1]. The *faction of overlapping documents* is simply the ratio of the number of common documents in the two rankings and the number of top k documents considered.

The features we use are simple aggregates of features that are already used for ranking and can be computed efficiently. The average feature similarity computation requires two passes over the feature sets of the top ranked documents for all rankers – one pass to compute the centroid and the other to compute the distance of each feature vector to the centroid. Compared to scoring and sorting a large number of documents, which web search engines typically do, the time taken to compute the aggregates and the average feature similarity for the top few documents can be relatively small.

In the subsequent sections, we present empirical evaluation of ReEff for both ranker selection and ranker fusion.

5.1.3 Experiments

We conduct ranker selection experiments to evaluate the utility of ReEff for selecting between rankers. We conduct these selection experiments on a large publicly available Microsoft Learning-to-Rank data set (MSLR-Web-10K) consisting of 10,000 queries. Each query-document pair in this dataset is represented by a set of 136 features, including low-level features such as covered query term ratio, individual retrieval models such as TF-IDF (Spärck-Jones, 1972), Okapi BM25 (Robertson et al., 1994), and variants of Language Modeling approaches (Ponte & Croft, 1998a) that

[1]This is akin to intra-cluster similarity used to measure the quality of a cluster of n-dimensional data points.

Table 5.2: Ranking Algorithms and their mean average precision on the MSLR Web 10K dataset.

Name	MAP	Description
BM25	0.2561	Single feature ranker (Robertson et al., 1994).
RankBoost	0.2770	Boosting algorithm that minimizes discordant pairs in ranking (Freund et al., 2003)
AdaRank	0.2840	Boosting algorithm that directly optimizes for MAP (Xu & Li, 2007).
L1-LogReg	0.3031	Logistic Regression with L1-constraints[3].
AFS	0.3053	Co-ordinate ascent-based algorithm with direct optimization for MAP (Metzler, 2007).

are applied to different fields such as URL, title, anchor and body and document quality features such as page rank[2].

We use a five-fold cross-validation approach for all our experiments. The training fold is used to train the ranking algorithms and the trained models are used to produce rankings on both the training and test set. We use five rankers listed in Table 5.2, including four competitive learning to rank methods and a single feature ranker BM25, which is the best individual feature. The table shows the MAP of the rankers on the entire data set and we see that the AFS based algorithm with direct optimization for MAP (Metzler, 2007) performs the best and BM25, the single feature ranker, performs the worst.

We use four baseline methods for ranker selection for comparison with *Difference*.

1. Prior-Based - Prior-Based selects a ranker using a random draw from a multinomial distribution which specifies the likelihood of each ranker being the best

[2]The data set and the full list of the features are available at: `http://research.microsoft.com/en-us/projects/mslr` (see Appendix for a brief listing of the features.)

[3]We use the R Lasso2 package available at `http://cran.r-project.org/web/packages/lasso2/index.html`

for a query. We estimate the parameters of this distribution from the training queries.

2. Best-on-Train - Instead of using the best individual ranker on the entire dataset, we use a stronger baseline, Best-on-Train. For each test fold, Best-on-Train selects the ranker that achieves the best average performance (in terms of MAP) on the corresponding training fold. We find that this leads to a better performance compared to only using AFS, the ranker with the best MAP on the entire dataset.

3. LTS (Peng et al., 2009) - LTS is a learning to select algorithm that utilizes the training data to estimate the performance of each ranker on queries that are similar to the test query. For each ranker R, LTS computes KL-divergences of the score distribution produced by R with respect to the score distributions produced by a baseline ranker such as BM25. This divergence feature is computed for the test query and the training queries. Then, LTS identifies from the training queries, the k-nearest neighbors (k-nn) for the test query using the divergence feature as the distance. The performance of R on this set of k-nn training queries is used as the expected performance of R on the test query. The process is repeated for all available rankers, and the ranker R^* which has the highest expected performance is selected as the best ranker for the test query.

   Peng et al. (2009) use BM25 as their base ranker [4] but in our experiments since we use BM25 as one of the candidate rankers, we use TF-IDF as the base ranking function. We use a held-out set in the training data to estimate the two parameters for the LTS approach, the number of nearest neigbors (k) and the number of top ranked documents (n) used to compute the score distributions.

[4]They do not use the base ranker as a candidate ranker available for selection but only use it to compute divergences.

4. *Independent* - *Independent* learns a Random Forest (Liaw & Wiener, 2002) based regression model that can predict the performance of each ranker *independently*. For each query, the ranker with the highest predicted performance is selected as the best ranker. *Independent* uses all the features listed in Table 5.1 except for the *overlapping documents* feature, which is a feature defined over two rankers. The features are extracted from the top 20 documents from each ranker[5].

5. *Difference* - *Difference* learns a Random Forest (Liaw & Wiener, 2002) based regression model that predicts the difference between a baseline ranker and the other alternate rankers. The alternate ranker with the highest positive difference is selected. If no candidate ranker has a positive predicted difference, then the baseline ranker is used. *Difference* uses the Best-on-Train ranker as the baseline ranker. The features used to learn the regression are extracted from the top 20 documents for each ranker.

6. Oracle - To demonstrate the best possible selection performance, we also include the selection performance that can be achieved by an oracle, i.e., a selection function that always selects the best ranker for each query.

5.1.4 Results

We compare ranker selection results in terms of mean average precision (MAP) of the selected rankings and selection accuracy. To compare selection accuracy, we designate the ranker chosen by Best-on-Train as the baseline ranker and treat the other rankers as alternate rankers. Note that Best-on-Train may be different for different folds. Then, we evaluate the selection methods on the number of queries for

[5]We experimented a range of settings [1,5,10,15,20,...,50] and chose 20, the setting that maximized the correlation of predicted effectiveness with MAP on the training data.

which they choose an alternate ranker, and the number of times the selected alternate ranker performed better, worse or the same when compared to the baseline ranker.

Table 5.3: Ranker selection results: Each fold comprises 2000 test queries. Bold-face indicates the best (non-oracle) performance in each column. * indicates statistically significant improvements over the Best-on-Train, determined using Fisher's randomization test ($p < 0.05$).

Method	**Fold 1**	**Fold 2**	**Fold 3**	**Fold 4**	**Fold 5**	**Average**
Best-on-Train	0.3056	0.3131	0.3027	0.3048	0.3111	0.3074
Prior-Based	0.2945	0.2918	0.2749	0.2922	0.3004	0.2908
LTS	0.3038	0.3023	0.3007	0.3034	0.3094	0.3039
Independent	0.3041	0.3095	0.3013	0.3038	0.3118	0.3061
Difference	**0.3131***	**0.3142**	**0.3071***	**0.3098***	**0.3189***	**0.3126***
Oracle	0.3658	0.3640	0.3604	0.3652	0.3727	0.3656

(a) Mean-average Precision for ranker selection.

Method	**Alt.Queries**	**Better (%)**	**Worse (%)**	**Same (%)**
Prior-Based	6969	2713 (38%)	4216 (62%)	0 (0%)
LTS	6323	2736 (43%)	3032 (48%)	555 (9%)
Independent	6646	2855 (42%)	3037 (46%)	754 (12%)
Difference	4506	2197 (49%)	1716 (38%)	513 (20%)

(b) Selection accuracy relative to the Best-on-Train baseline on the entire set of 10,000 queries. Better, Worse, and Same indicate the number of queries for which the method was better, worse or same compared to Best-on-Train.

Table 5.3 shows the ranker selection results in terms of mean average precision (MAP) on 10,000 queries (top) and selection accuracy (bottom). The results show that *Difference* outperforms all selection baselines. Below, we present a detailed analysis of the selection results.

Baselines.

Best-on-Train selects AFS as the best ranker for three folds and Logistic Regression for the other two folds. For each fold, the best performing ranker on the training queries also turns out to be the best on test queries. Note that Best-on-Train is a stronger baseline than the best individual ranker, as it performs better than only using AFS, the ranker with the best MAP (0.3053) on the entire data set.

Compared to this stable Best-on-Train baseline, Prior-Based random selection performs worse, which shows that knowing the prior distributions of the performance of different rankers alone is not adequate for the selection task.

Using LTS for ranker selection also performs worse than the baseline Best-on-Train. LTS selects alternate rankers for a large number of the queries (for more than 63% of the queries) but mostly unsuccessfully – selection using LTS leads to worse performance in 48% of the queries, while only providing improvements for 42%.

Prior work (Peng et al., 2009) has shown that LTS can provide substantial improvements for the task of selecting between three rankers on Letor 3.0, a smaller learning-to-rank data set (Liu et al., 2007). To validate our implementation of LTS, we conducted selection experiments on the 125 Topic distillation queries of the Letor 3.0 data set. The features used in Letor 3.0 are a subset of the features used in the MSLR-Web-10K data set that we use for our experiments. On this smaller data set, but for the same set of rankers, LTS did achieve improvements in MAP over the Best-on-Train[6]. However, on the larger web data set, despite conducting an exhaustive grid search over a wide-range of values for the k and n parameters (the number of nearest neighbors and the number of top-ranked documents respectively), LTS does not provide any improvements[7]. Moreover, we find that the overall correlation between the expected performance of each ranker, determined as the average performance of the k-nearest neighbors, and the actual performance on the test query is very low (Pearson's ρ of 0.11 for LTS, compared to 0.27 for *Difference*), which in part explains the overall poor selection performance.

---

[6]LTS improved MAP by 0.0053 over the Best-on-Train MAP of 0.2206, while *Difference* gave an improvement of over 0.0109 in MAP.

[7]The range for k was [1,2,...5,10,15,20,..,50,100,200,...500,1000] and the range for n was [1,2,...,10,20,...50,100]. The best k was 500 for four folds, and 400 for the other and the best n was 5 for all five folds.

Independent also performs worse compared to Best-on-Train both in terms of MAP and in terms of the positive to negative impact. *Independent* hurts more queries than it improves, as nearly 46% of the alternate ranker selection leads to worse performance, whereas less than 42% of alternate ranker selections leads to improvements. Despite having access to all the base set of features that *Difference* utilizes, *Independent* does not provide improvements over Best-on-Train, whereas *Difference* performs consistently better. This shows that when using ranker based and retrieval based features directly modeling relative effectiveness is more useful than independently estimating the effectiveness of each ranker.

Difference

Difference provides consistent average improvements in MAP. *Difference* performs the best on each individual fold and its improvements over Best-on-Train are statistically significant in all but one fold. ReEff achieves about 9% of the possible oracle improvements, the improvements that can be achieved with a selection oracle that selects the best ranker for every query (last row in Table 5.3). Even though the average improvements on the entire data set are small (about 0.0052 in absolute MAP), the actual gains achieved by *Difference* are substantial improvements (0.0113 in absolute MAP) obtained on a smaller subset of queries, the subset for which *Difference* selects alternate rankers. The average relative improvement within this subset of queries is roughly 4%.

Further, *Difference* also performs better than the other baselines in terms of selection acuracy. Since *Difference* selects alternate rankers for fewer queries (less than 50%), it provides better performance for fewer queries, compared to the other baselines. However, in 49% of the cases when *Difference* selects an alternate ranker, it results in an improvement over the baseline, whereas only 38% of the times leads to a decrease in performance. We believe that this positive to negative impact ra-

Table 5.4: Fusion versus Selection. Comparison of fusion techniques against selecting the best ranker using *Difference*. Mean(ΔAP) denotes the mean of differences in AP between the baseline Best-on-Train and corresponding fusion technique. b,r,c and m superscripts indicate statistically significant improvements (determined using Fisher's randomization test with $p < 0.05$) over the Best-on-Train baseline, Reciprocal Rank (RR), CombMNZ (CM), and MapFuse(MF) methods respectively.

Method	**MAP**	**Mean(ΔAP)**	**Better**	**Worse**	**RI**
Best-on-Train	0.3074	+0.0000	0	0	NA
RR	0.3017	-0.0058	4062	4874	-0.080
CM	0.3108	$+0.0031^{b}$	4637	4292	+0.034
MF	0.3052	-0.0022	4229	4706	-0.048
Difference	0.3126	$+0.0052^{b,r,c,m}$	2196	1718	+0.048

tio is a key strength of *Difference* and we provide further analysis on this aspect of *Difference*'s performance in Section 3.2.4.

In summary, the ranker selection experiments show that using *Difference* to select the best ranker for each query can outperform using a fixed ranker for all queries. In the next section, we explore the utility of *Difference* for improving ranker fusion.

5.2 Ranker Fusion

In addition to using *Difference* for selecting the best ranker for a given query, we also explore the use of *Difference* for improving ranker fusion – combining the rankings from multiple rankers.

First, we conduct ranker fusion experiments to compare the benefits of selection and fusion. We use three fusion techniques: 1) Reciprocal Rank - a rank based technique, 2) CombMNZ - a score based technique and 3) MapFuse - a rank based technique that utilizes past performance of rankers to perform weighted combination.

- *Reciprocal Rank* (Cormack et al., 2009) (RR) uses the rank information of documents in each ranking to produce fused results. Reciprocal Rank is a competitive rank based fusion algorithm shown to achieve good fusion performance on the Letor

datasets (Liu et al., 2007). Given a set of rankers $\mathbf{R}$, the final Reciprocal Rank score of a document d for query q is computed as follows:

$$RR(q,d) = \sum_{R \in \mathbf{R}} \frac{1}{k + rank_R(q,d)}$$

where, $rank_R(q,d)$ is the rank of document d in R's ranking for query q. k is a free parameter, which we set to 60 based on training set performance.

- *CombMNZ* (Lee, 1997) (CM) uses the sum of normalized scores assigned by the rankers, which is then weighted by the number of rankings in which the document was retrieved in the top k ranks. The final score is computed as follows:

$$CM(q,d) = |M| \sum_{R \in \mathbf{R}} nscore_R(q,d)$$

where, $M = \{R \in \mathbf{R} | rank_R(q,d) \leq k\}$ and $nscore_R(q,d)$ is the min-max normalized score assigned to document d by R for query q. We set k to 1000 based on training set performance.

- *MAPFuse* (Lillis et al., 2010) (MF) uses the performance of each ranker on the training set of queries to produce the final score. The MAPFuse score for each document is computed as follows:

$$MF(q,d) = \sum_{R \in \mathbf{R}} \frac{MAP_R(q)}{rank_R(q,d)}$$

where, $MAP_R(q)$ is the mean-average precision of the ranking produced by ranker R for query q.

Table 5.4 shows results for fusion using all rankers and for selecting the best ranker for each query (selection) . For each method, we tabulate 1) the mean of differences in

AP with respect to the Best-on-Train baseline, denoted as Mean(ΔAP), 2) the number of queries for which the technique was better than Best-on-Train, 3) the number of queries for which the technique was worse, and 4) the Robustness Index (Collins-Thompson, 2009), defined as RI = (# Better − # Worse)/n, where n is the total number of queries.

Using *Difference* to select the best ranker for each query is better than fusing the results from all five rankers. In fact, only one fusion technique, CM, provides additional average improvements over the Best-on-Train baseline. Also, selection performance is slightly better in terms of robustness measured by RI. Even though CM provides improvements for a larger proportion of queries compared to selection (46% versus 22%), it also degrades performance for a larger proportion (42% versus 17%). The trend is similar for the other two fusion techniques as well.

These results suggest that selection and fusion provide different benefits. Also, selection is better for some queries, while fusion is better for others. Therefore, we explore the use of *Difference* to augment fusion in a query-dependent fashion.

5.2.1 Approach

Difference provides estimates of the relative differences of the alternate rankers with the baseline ranker for each query. We utilize these estimated relative differences to improve fusion in three ways.

1. Selective Fusion - We target selective fusion – selecting queries for which fusing the rankings can be beneficial. Our initial analysis showed that for some queries, fusion is more effective than selection, whereas for others selection can be more effective. We model selective fusion as the task of choosing between the individual rankings and the fused ranking. The best ranking (individual or fused) is selected based on the estimated relative differences.

2. Selecting Rankers - Selecting the most effective rankers per query can help improve fusion performance. In many cases, we find that fusing fewer but more effective rankings is better than fusing all available rankings. We utilize the estimated relative effectiveness of the rankings to induce an ordering on all the available rankers and select the top k rankings to fuse for each query.

3. Weighting Rankers - Using weights that reflect the relative quality of the rankings that are being fused can also help improve fusion performance: intuitively, documents present in highly effective rankings should be preferred to documents from less effective rankings. However, since the effectiveness of the rankings is not known for all queries, we use the relative differences that *Difference* estimates as weights indicating the effectiveness of the rankers. We compare this approach to MAPFuse (Lillis et al., 2010), a recently proposed technique for utilizing the performance of the rankers on the training set as weights for combining rankings.

5.2.2 Experiments

We use ReEff to augment the fusion techniques in three ways.

- *Selective Fusion* (Selective {RR, CM, MF}) - First, we use *Difference* to select between the individual rankings and the fused ranking. The fused ranking is generated by combining all the available individual rankings. In this setting, the baseline ranker is the Best-on-Train ranker, which is chosen from the set that includes both the individual rankers as well as the fusion ranker.

- *Ranker Selection* (S+RR, S+CM, S+MF) - Second, we use *Difference* to select the top K rankers for fusion. We report the performance of fusing top 2 to top 5 (all) rankers using each of the three fusion techniques.

- *Ranker Weighting* (W+RR, W+CM,W+MF) - Third, we use *Difference* to assign weights to the rankings produced by each ranker. First, we use *Difference* to obtain predicted differences of the alternate rankers with respect to the baseline ranker. Then, we normalize the predicted differences using a min-max normalization to avoid negative weights[8]. Finally, the document scores for each ranker are multiplied by the corresponding normalized weight and these weighted document scores are used by the fusion techniques to produce the final fused ranking. We report the performance of weighting in conjunction with ranker selection[9].

  The final score of a document for the fusion techniques is computed as follows:

  1. Weighted Reciprocal Rank (W+RR)

$$W{+}RR(q,d) = \sum_{R \in \mathbf{R}} \frac{w_R}{k + rank_R(q,d)}$$

  2. Weighted CombMNZ (W+CM)

$$W{+}CM(q,d) = \sum_{R \in \mathbf{R}} w_R \ \times nscore_R(q,d)$$

  3. Weighted MAPFuse (W+MF)

$$W{+}MF(q,d) = \sum_{R \in \mathbf{R}} \frac{w_R}{rank_R(q,d)}$$

  where, $rank_R(q,d)$ is the rank of the document in R's ranking, $nscore_R(q,d)$ is the min-max normalized score of document d assigned by R and w_R is the normalized ReEff weight for R.

---

[8]The baseline ranker's predicted difference is set to zero.

[9]The impact of weighting alone can be compared when fusing all available rankers.

Table 5.5: Selective Fusion: Results of selective fusion. Mean(ΔAP) denotes the mean of differences in AP between the baseline Best-on-Trainand corresponding fusion technique. b,r,c,m, and e superscripts indicate statistically significant improvements (determined using Fisher's randomization test with $p < 0.05$) over the Best-on-Train baseline, Reciprocal Rank (RR), CombMNZ (CM), MapFuse(MF), and *Difference* methods respectively. Bold-face entry indicates the best MAP.

Method	**MAP**	**Mean(ΔAP)**	**Better**	**Worse**	**RI**
Best-on-Train	0.3074	+0.0000	0	0	NA
RR	0.3017	-0.0058	4062	4874	-0.080
CM	0.3108	$+0.0031^{b}$	4637	4292	+0.034
MF	0.3052	-0.0022	4229	4706	-0.048
Difference	0.3126	$+0.0052^{b,c}$	2196	1718	+0.048
Selective RR	0.3141	$+0.0060^{b,r,e}$	2676	2076	+0.060
Selective CM	**0.3156**	$+0.0081^{b,c,e}$	3550	2654	+0.090
Selective MF	0.3132	$+0.0058^{b,m}$	3082	2494	+0.059

5.2.3 Results

5.2.3.1 Selective Fusion

Table 5.5 shows the results for 1) fusion – fusing the results of all rankings using RR, CM, and MF, 2) ranker selection – selecting the best individual ranking for each query, shown as (*Difference*), and 3) selective fusion – selecting between the individual rankings and the fused ranking (Selective RR, Selective CM and Selective MF).

For all three fusion techniques, selective fusion performs better than fusion. All improvements of selective fusion over the corresponding fusion methods are statistically significant. Furthermore, selective fusion also performs better than ranker selection and the improvements over ranker selection are statistically significant for RR and CM.

Selective fusion combines the merits of both fusion and selection. For example, when using CM, selective fusion increases the number of queries with a positive impact by about 13% compared to selection, while also increasing the number of queries with negative impact by about 9%. This trade-off leads to an overall improvement in MAP, and also provides substantial improvements in overall robustness as shown by the RI

Table 5.6: Selecting and Weighting Rankers for Fusion: Mean average precision (MAP) results for selecting and weighting rankers using *Difference*. Bold-face entries indicate the best performing method for each column. * indicates statistically significant improvements of the S+ or W+ methods over the corresponding top K ranker fusion (B+ methods). ** indicates statistically significant improvements of the weighted fusion (W+ methods) over the corresponding selection of rankers (S+ methods) using *Difference*. All statistical significances are determined using Fisher's randomization test (p-value < 0.05).

Method	Top 1	Top 2	Top 3	Top 4	All
B+RR	0.3074	0.3099	0.3098	0.3052	0.3017
B+CM	0.3074	0.3179	0.3180	0.3126	0.3108
B+MF	0.3074	0.3153	0.3150	0.3112	0.3061
S+RR	**0.3126***	0.3130*	0.3115*	0.3077*	0.3017
S+CM	**0.3126***	0.3187	0.3195*	0.3154*	0.3108
S+MF	**0.3126***	0.3164*	0.3168*	0.3135*	0.3061
W+RR	**0.3126**	0.3133	0.3123**	0.3100**	0.3067**
W+CM	**0.3126***	**0.3189**	**0.3202****	**0.3174****	**0.3153****
W+MF	**0.3126***	0.3167*	0.3173*	0.3156**	0.3112**

values. These results suggest that selective fusion can help to combine the benefits of both fusion and selection.

5.2.3.2 Selecting Rankers for Fusion

Table 5.6 shows the performance of selecting the top k rankers for fusion. The rows for the B+ and S+ methods in Table 5.6 show the results of using Best-on-Train and *Difference* respectively for selecting the top k rankers. The Top 1 column corresponds to the case of selecting a single ranker for each query, whereas the Top 5 column corresponds to the case of using all the five rankers for fusion. The B+ entries for Top 1 show the baseline performance for Best-on-Train whereas the S+ and W+ entries show the performance of selecting using *Difference*.

For all three fusion techniques, the performance of fusing the rankings of the top few rankers is better than fusing all rankings. As shown by the B+ rows in the table,

when using Best-on-Train to select the top few rankers, using just the top 2 or 3 rankers provides the best performance.

Using *Difference* to select rankers yields substantial improvements over using Best-on-Train to select rankers. All corresponding improvements, except for selecting the top 2 rankers for CM fusion, are statistically significant. Further, the best performance with selection is achieved by S+CM for Top 3 i.e., when selecting the top 3 rankers for CM (MAP 0.3195) . This setting is also significantly better than the best performance that can be achieved by selecting rankers using Best-on-Train – when using the top 2 rankers (MAP 0.3179). These results suggest that ranker selection using *Difference* can further improve fusion performance.

5.2.3.3 Weighting Rankers for Fusion

The W+ rows in Table 5.6 show the results for using the relative difference weights in conjunction with ranker selection. Using *Difference* for weighting rankers yields further improvements in all cases. When fusing the top 2 rankers, weighting the selected rankers does not yield substantial improvements. However, when fusing the top 3, 4 and 5 rankers, weighting provides substantial additional improvements for all fusion techniques. RR and CM do not use any weights on the rankers, and by introducing some weighting on the rankers through *Difference* we obtain additional improvements. However, it is worth noting that MF already uses weights on the rankers and replacing these static weights with query-dependent weights from *Difference* provides substantial additional improvements. This shows that *Difference* yields reliable query-dependent weights that can be used for improving fusion.

In summary, we find that fusion and selection provide different types of benefits and using *Difference* we can improve the combination of multiple rankers through selective fusion, ranker selection and ranker weighting.

5.3 Analysis

We analyze the performance of *Difference* to identify the features that are most important for selection and also to better understand the types of improvements that *Difference* provides.

5.3.1 Feature Importance

Table 5.7 shows the list of the most important features for selection, where importance is determined as the normalized reduction in the random forest regression error. The most important features include a mix of the ranker based and retrieval based features – the ranker scores and their aggregates, aggregates of the retrieval features such as click-based and BM25 features, as well as the average feature similarity measure. For ranker scores the most important aggregates are those that characterize the variance in the ranker scores and the two additional aggregates, skew and kurtosis, which characterize the shape of the score distributions. The most important retrieval based features correspond to aggregates of the query-URL click count, which is a strong indicator of document relevance. The average feature similarity feature, which measures similarity of the top ranked documents is also one of the top 10 important features. Individual retrieval model scores such as BM25, language modeling scores – which are strong indicators of relevance – also turn out to be important features for modeling relative effectiveness.

To better understand the impact of ranker scores versus retrieval-based features, we also conduct selection experiments with each individual group of features.

Table 5.8 shows the selection performance of the two groups on a single fold (Fold 1) of data. We compare the feature groups in terms of MAP, improvements over Best-on-Train, shown as ΔAP, and the robustness index (RI). Using ranker score based features alone does not yield any improvements over the baseline and has poor robustness. Using retrieval based features alone provides good improvements

Table 5.7: Feature Importance for *Difference* randomForest regression: Top ranked features sorted by the mean decrease in node purity, which is a measure of importance of the feature in the regression.

Feature Type	Aggregate Type	Importance
Ranker scores	standard deviation	1.66
Ranker scores	skew	1.59
Ranker scores	variance	1.57
Query-URL click count	geometric mean	1.42
Ranker scores	co-effic. of dispersion	1.37
Query-URL click count	mean	1.35
Ranker scores	kurtosis	1.29
Query-URL click count	variance	1.23
Average feature similarity	individual	1.18
Ranker scores	position 20	1.08
Query-URL click count	standard deviation	1.06
Ranker score	positions 1:19	1.02-0.87
Ranker scores	geometric mean	0.83
Ranker scores	arithmetic mean	0.80
BM25 body	max	0.71
BM25 whole document	max	0.67
Page Rank	max	0.67
Ranker scores	harmonic mean	0.64
...	...	...
LMIR.ABS whole document	variance	0.63
LMIR.JM whole document	variance	0.62
URL dwell time	geometric mean	0.61
Document length	max	0.61
URL length	co-effic. of dispersion	0.60

over the baseline. However, the combination of the ranker scores with the retrieval features yields the best performance, which suggests that these features carry different types of information. Further, it is interesting to note that the ranker scores are important features in the combined regression but they do not perform better on their own. There are fewer ranker score based features, when compared to retrieval based features. As a result, even though ranker scores turn out to be the most error-reducing features in the combined regression, there aren't enough of them to provide improvements on their own.

These results suggest that the selection performance of *Difference* depends on both retrieval based features as well as ranker score based features.

5.3.2 Distribution of Selection Gains

Figure 5.1 plots the selection gains against the oracle gains – gains that can be achieved if we have a perfect selection technique. Whenever there is high potential,

Table 5.8: Selection performance of different feature groups on a single fold (Fold 1). Ranker and Retrieval rows indicate performance of ranker score based, and retrieval-based features respectively.

Group	MAP	Mean(ΔAP)	Better	Worse	RI
Best-on-Train	0.3056	NA	NA	NA	NA
Ranker	0.3061	0.0005	609	590	0.01
Retrieval	0.3096	0.0037	412	353	0.03
All	0.3131	0.0075	494	405	0.05

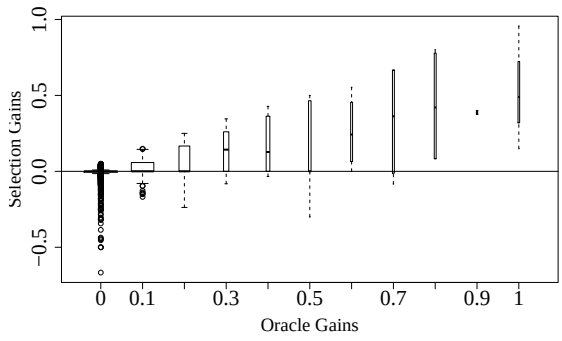
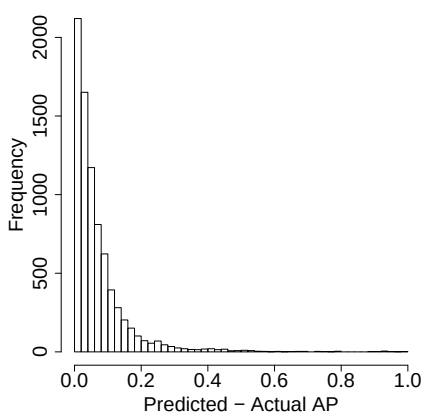


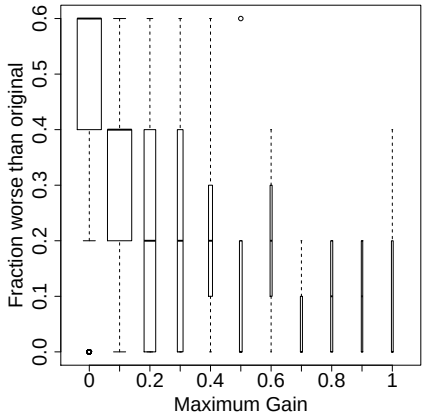
Figure 5.1: Oracle Performance versus Selection Gains: Distribution of selection gains (y-axis) against the gains that are possible using an oracle (x-axis).

Difference provides gains in most cases and most of the errors in selection happen in cases where the potential is low. This shows that *Difference* is effective at modeling large positive differences more effectively. Similar to our observations with query reduction, we find that most prediction errors for *Difference* are small (within 0.2 in average precision (AP)) as shown in Figure 5.2(a)). Also, for queries with large potential, the selection problem becomes easier. As shown in Figure 5.2(b) the fraction of alternatives that are worse than the original drops for queries with large potential.

We find that selection is more likely to be helpful for queries whose baseline performance is poor. Figure 5.3 shows the distribution of large differences in AP achieved by using *Difference* against the performance of the baseline ranker, Best-on-Train. The boxplot shows the distribution of differences in AP that are greater than 0.1 (we have 1875 such instances using *Difference*). Large positive differences



(a) Prediction errors: Frequency histogram of difference between predicted and actual



(b) Selection difficulty: Boxplot showing distribution of alternatives that are worse than the baseline (y-axis) against the maximum possible gains (x-axis)

Figure 5.2: Prediction Errors and Selection difficulty.

are obtained for queries whose performance on the baseline ranker is below 0.4 in AP and large negative differences are obtained for queries whose baseline performance is above 0.4 in MAP. This is in part because the potential for gains are higher when the baseline ranker performs poorly.

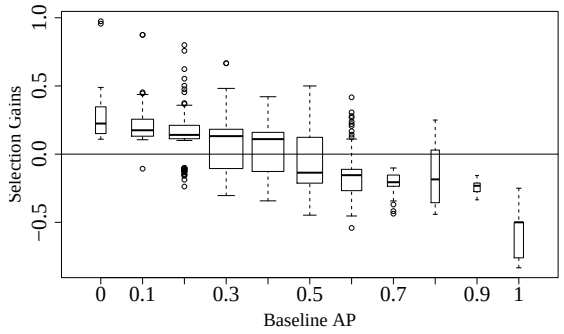
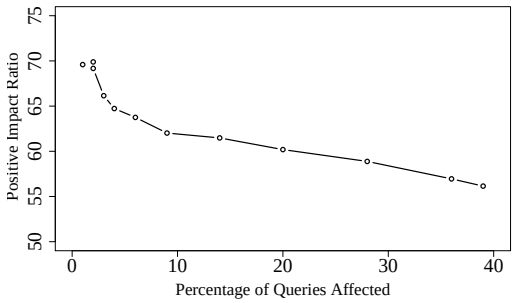


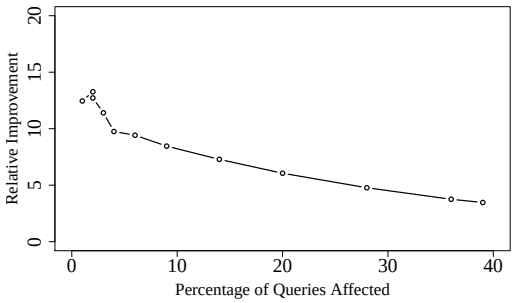
Figure 5.3: Baseline Performance versus Selection Gains: Distribution of differences in AP that are larger than 0.1.

5.3.3 Quality of Impact

Next, we analyze the quality of impact of ranker selection with respect to the number of queries affected. To this end, we conduct selection experiments where we impose a threshold on the predicted difference – i.e., we select an alternate ranker only if the predicted relative difference exceeds the specified threshold. When the threshold is set to zero, it is equivalent to the selection results we report in Section 5.1.3. However, as we set the threshold to increasing positive values, we select alternate rankers for fewer queries and consequently ranker selection affects fewer queries. For the purposes of this analysis, we experiment with thresholds between 0.0 and 0.05 with increments of 0.005.



(a) Positive Impact on Affected Queries: The positive impact ratio – the ratio of improved queries to total queries – plotted against the percentage of total queries that are affected.



(b) Relative Improvement on Affected Queries: The percentage of relative improvement over the Best-on-Train baseline on the subset of queries that are affected.

Figure 5.4: Trade-offs in Selection Impact versus Percentage Queries Affected.

Figure 5.4(a) shows the effect of thresholding on the *positive impact ratio* – i.e., the ratio of number of queries with a positive impact to the total number of queries with a non-zero impact. When fewer queries are affected the positive impact ratio improves. Reducing the affected queries from 40% to 10% leads to only a small improvement in positive impact ratio (about 5%), but further reductions in the percentage of affected queries leads to dramatic improvements in positive impact. When only affecting around 3% of the queries, selection can have a positive impact of about 70%. Figure 5.4(b) shows similar trends of the magnitude of relative improvements

over the Best-on-Train on the set of queries that are affected. When reducing the percentage of queries affected, we see higher relative improvements leading up to nearly 13% increase on a subset of 5% of the total queries. These results show the potential for calibrating ranker selection using ReEff to achieve a desired trade-off in percentage queries affected versus quality of impact.

5.3.4 Impact of Rankers

Table 5.9 shows the selection performance as the number of available alternate rankers is increased. As shown by entries in the oracle column, as the number of rankers is increased, the potential value for selection increases, with most potential delivered by the top 2 rankers, while adding more rankers leads to smaller diminishing increases. Accordingly, we see that most gains are achieved by selecting between the top 2 rankers (more than 80% of the total gains), and adding more rankers provides a smaller additional increase (less than 20% of the total gains). Importantly, we see that *Difference* utilizes all available rankers to deliver improvements over the baseline ranker.

Table 5.9: Ranker Selection results on a single fold (Fold 1) while increasing number of rankers for *Difference*.

Rankers	**ReEff**	**Oracle**
1	0.3056	0.3056
2	0.3117	0.3405
3	0.3121	0.3512
4	0.3121	0.3592
5	0.3131	0.3658

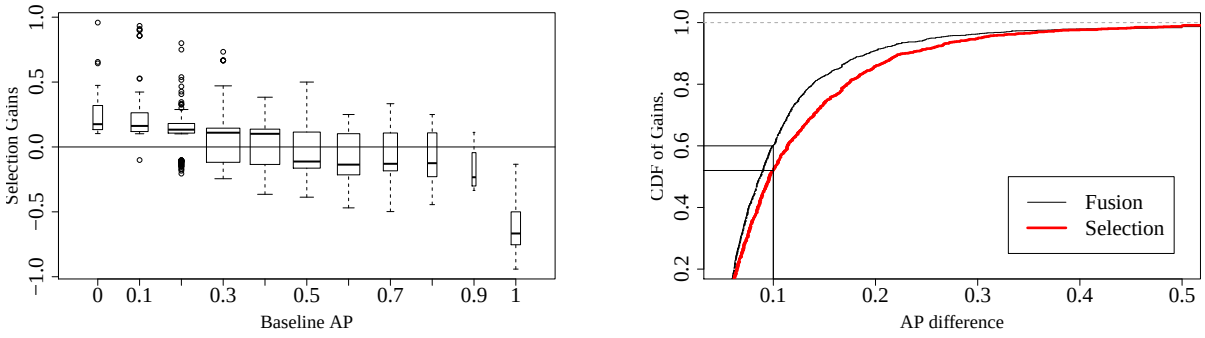
In addition to using the best individual ranker (Best-on-Train) as the baseline ranker, we also conduct experiments using other rankers as the baseline ranker on a single fold (Fold 1). Table 5.10 shows the performance of using other rankers as the baseline ranker for *Difference*. In all cases, we find that selection improves over the corresponding baseline. More importantly, selection always improves over the

best individual ranker Logistic Regression for this fold. This shows that while using the best individual ranker as the baseline yields substantial improvements, selection performance is not entirely due to the choice of the baseline ranker alone.

Table 5.10: Selection using other rankers as baselines on Fold 1. * indicates statistically significant improvements over the Best-on-Train baseline, when significance is determined using a Fisher's randomization test with $p < 0.05$.

Baseline	MAP(Baseline)	MAP(ReEff)
AdaRank	0.3018	0.3106*
BM25	0.2548	0.3113*
AFS	0.3005	0.3099*
RankBoost	0.2790	0.3118*
LogReg	0.3056	0.3131*

5.3.5 Fusion Analysis



(a) Distribution of fusion gains versus Best-on-Train AP.

(b) Positive improvements of fusion and selection.

Figure 5.5: Distribution of fusion improvements.

Figure 5.5(a) shows the distribution of large differences in AP achieved by using CM fusion using all five rankers plotted against the performance of the baseline ranker, Best-on-Train. The boxplot shows the distribution of absolute differences in AP that are greater than 0.1. Similar to the performance of selection, fusion yields improvements for queries with poor baseline performance, and degrades performance of queries with higher baseline performance. However, when compared to selection (in Figure 5.3), fusion yields improvements for more queries that have higher baseline performance.

Figure 5.5(b) shows the distribution of positive improvements over Best-on-Train for fusion and selection. The thinner upper curve represents the cumulative density function (CDF) for fusion's improvements over Best-on-Train and the thicker lower curve represents the CDF for selection. The CDF's show that selection typically provides larger improvements compared to fusion. For example, more than 50% of selection's improvements are more than 0.10 in MAP, whereas only 40% of fusion's improvements are greater than 0.10 in MAP. However, as we look at larger improvements – for example for improvements of over 0.3 in MAP – there is no clear trend. This is in part because there are fewer queries for which such large improvements are obtained either through fusion or through selection. In conjunction with the results from Table 5.4, the distribution of these large improvements shows that fusion and selection provide different types of improvements. Fusion provides smaller improvements over a large subset of queries, whereas selection provides larger improvements over a smaller subset.

5.4 Summary

In this work, we showed that the *Difference* formulation is effective for selecting between five different ranking algorithms on a large learning-to-rank data set. It provides small but significant improvements over using a fixed ranker: *Difference* obtains about 3.75% relative gains averaged over 45% of the queries and up to 13% on a subset of 5% of the queries.

As with query reduction using ReEff, we find that the *Independent* formulation is not as effective as *Difference*. Our experiments with fusion show that selection and fusion yield different types of benefits: Selection yields large gains on a smaller set of queries, whereas fusion yields smaller gains over a larger set. Query-dependent fusion using *Difference* showed that three state-of-the-art fusion techniques can be

improved through selective fusion, selecting a set of rankers for fusion, and by using the predicted differences as weights for fusion.

Our analysis of the selection results show that *Difference* utilizes both ranker-based and retrieval-based features to effectively select between rankers, provides substantial gains when there is large potential, and is amenable to thresholding as a technique for controlling the quality of its impact.

CHAPTER 6

EFFICIENCY

Efficiency or cost constraints can limit using all alternatives for all queries. We consider a specific setting, the *black-box meta-search* scenario. Meta-search engines combine results from multiple web search engines (e.g., Dogpile, MetaCrawler, Ixquick). The individual web search engines are used as black-box systems with no access to the internals such as scores, features or indexes. In this setting, accessing certain search engines (rankers) can be expensive due to commercial licensing (Musser, 2007). In some cases, accessing all the rankers can also be *time-consuming*. For instance, one or more rankers may have high latencies, thus hurting the overall response times. Therefore, using all rankers for all queries can be inefficient. In this setting, we seek to improve efficiency by minimizing the number of times additional alternatives are queried, while also seeking to maximize effectiveness by leveraging multiple rankers.

We consider a *learning to query* problem - the problem of deciding when to query alternate rankers by inspecting the results of a single baseline ranker alone. For evaluation, we model the competing efficiency and effectiveness goals by using a combined measure, which can be adjusted to reflect different trade-offs.

To address the learning to query problem, we build a threshold-based classifier that can be trained to optimize for the combined measure. Most search engines do not provide access to the ranking scores or internal features that they use to rank documents, so we cannot readily use ReEff's features which rely on retrieval scores and retrieval features. Instead, we develop features based solely on the query itself, and the results page of a baseline ranker.

We conduct experiments on the TREC Gov2 collections to demonstrate the viability of this approach. When selecting between two rankers, we find that compared to a simple prior-based approach the thrsehold-based approach yields up to about 7% improvement on the combined measure for effectiveness and efficiency. We also develop a technique for augmenting the training data using automatically generated overlap-based data and show that this yields further improvements of up to 15%.

In this chapter, we formally describe the problem, present our approach for predicting when to query an alternate ranker and discuss experimental evidence that shows promise for the proposed techniques.

6.1 Approach

We consider the standard meta-search scenario, where the meta search engine has access to *rankers* that retrieve and rank documents from different (but potentially overlapping) collections. Typically, a meta-search engine *always* queries *all* the rankers and fuses the returned results (Aslam & Montague, 2001; Selberg & Etzioni, 1995). Alternatively, as we showed in Chapter 5 we can use ReEff to select a query-dependent subset of rankers to improve fusion effectiveness. However, in order to select a subset the approaches, ReEff uses features based on the results that are obtained from querying all the rankers. This cost of querying all the rankers for all queries is what we seek minimize.

Instead of always accessing all the rankers, we propose a more cost-effective approach that always uses a single base ranker, and only accesses additional ranker(s) when their expected utility exceeds a specified threshold. We define the utility in terms of the additional relevance information that can be obtained from the additional rankers. Beitzel et al. (2003, 2004) show that when the retrieval systems used

are highly effective but have systemic differences[1], the improvements due to fusion are largely due to improvements in recall due to addition of new relevant documents. In the web meta-search scenario that we consider, the rankers are typically both effective and diverse, combining search results with low overlap is are more likely to produce better fused results compared to search results with higher overlap (Dogpile.com, 2007). Therefore, we define a relevance-based effectiveness metric that measures the fraction of new relevant documents that can be obtained by querying the candidate ranker.

In this section, we formally define the *learning to query* (LTQ) problem, develop an evaluation measure, and describe a threshold-based classification approach using easy-to-compute features.

6.1.1 Problem Definition

We make the following simplifying assumptions in defining the *learning to query* problem:

1. The meta-search engine has access to two rankers: (i) a base ranker, B and (ii) a candidate ranker, C. The base ranker, B, is always queried. The candidate ranker, C, is queried only if some criteria are met. Note that while conceptually simple, this setting can be easily extended. For instance, we can extend it to a case of multiple rankers by treating C as a set of candidate rankers.

2. We assume a *black-box* scenario, typical for meta-search engines on the web, in which the search engine has no access to the internal workings of its rankers such as their retrieval algorithms or their indexes. Instead, the meta-search engine can only submit a query q to a ranker R, and get in response a ranking $\mathbf{D}_R$, containing K retrieved results. Ranking $\mathbf{D}_R$ typically includes an ordered

[1]Systemic differences are not just differences in retrieval models but also include different tokenization, stemming, stop words etc.

list of links to retrieved documents (URLs in web search), each accompanied by a brief snippet that provides a short query-biased preview of the document content.

3. We assume a general cost setting, where every time we use the candidate ranker C, we incur a fixed (unit) cost rather than choosing a specific cost setting such as financial costs imposed by licensing restrictions or network costs, w

In this setting, the *learning to query* problem is to learn a decision function $\mathcal{I}$, which given a query q, and the results of the base ranker, B, indicates whether the candidate ranker, C should be queried as well to obtain additional results. The main goal of LTQ is to choose a decision function that balances the effectiveness and efficiency goals. In terms of effectiveness, we want to maximize the relevance gains, the amount of additional *relevant* information that can be obtained by querying the candidate ranker. In terms of efficiency, we want to minimize the number of times the candidate ranker is queried.

6.1.2 Evaluation Measure

We evaluate the decision function for LTQ using a measure that combines the effectiveness and efficiency gains.

Formally, let $\mathbf{Q}$ be a set of queries over which we measure the trade-off in effectiveness versus efficiency and let $\mathcal{I}(q,\mathbf{D}_B)$, be the indicator function which indicates whether, given a query q and a base ranker B, we query the candidate ranker C. Using this notation, we first define the effectiveness, and efficiency measures, which we then use to define the combined measure.

Effectiveness: The effectiveness of the indicator function, effect($\mathcal{I}$), is measured as the fraction of possible gains in relevance that are achieved when querying the candidate ranker in accordance with $\mathcal{I}$. Letting $\mathcal{G}(q)$ denote the relevance gain obtained

when querying the candidate ranker for query q, we define the effectiveness of $\mathcal{I}$ as follows:

$$\text{effect}(\mathcal{I}) = \frac{\sum_{q \in \mathbf{Q}} \mathcal{I}(q, \mathbf{D}_B) \times \mathcal{G}(q)}{\sum_{q \in \mathbf{Q}} \mathcal{G}(q)} \tag{6.1}$$

We define $\mathcal{G}(q)$, the gains obtained when using the candidate ranker, in terms of the new relevant documents retrieved by the candidate ranker C. As we mentioned earlier, in the meta-search scenario the rankers are typically both effective and diverse and in such cases a higher overlap in search results is less likely to yield improvements compared to a lower overlap (Beitzel et al. 2003,2004; Dogpile.com 2007).

Let $\mathbf{R}_B$ and $\mathbf{R}_C$ denote the sets of relevant documents in the top K results retrieved by $\mathbf{D}_B$ and $\mathbf{D}_C$, respectively. Then, we define relevance gain for query q as:

$$\mathcal{G}(q) = \frac{\min\Big(|\mathbf{R}_C \backslash \mathbf{R}_B|, K - |\mathbf{R}_B|\Big)}{K} \tag{6.2}$$

Thus, $\mathcal{G}(q)$ measures the number of additional relevant documents that can be added to the top K ranks of the baseline ranking, if we had an optimal combination algorithm. If the baseline ranking is already optimal (i.e., all top K ranks in the baseline were relevant), then querying the candidate ranker C cannot yield any additional improvements to the top K ranks. On the other hand, if the baseline ranking is not optimal, then C can contribute new relevant documents through combination.

Note that $\mathcal{G}(q)$ represents the *optimal* bound (in terms of *prec*@K) on the relevance gain from combining the rankings $\mathbf{D}_B$ and $\mathbf{D}_C$. Actual combination of $\mathbf{D}_B$ and $\mathbf{D}_C$ using existing techniques such as CombMNZ (Fox & Shaw, 1994), Bordafuse (Aslam & Montague, 2001) or probFuse (Lillis et al., 2006) may not achieve this bound, but – as previous work indicates (Beitzel et al., 2004) – their performance is likely to correlate with it.

Efficiency: According to our definition of the LTQ problem, every time the candidate ranker C is queried, we incur a fixed unit cost (as per the third assumption

in Section 6.1.1). This suggests a straightforward definition of efficiency gains as the proportion of times we *avoid* querying the candidate ranker when using $\mathcal{I}$. Formally, effic($\mathcal{I}$) is defined as follows:

$$\text{effic}(\mathcal{I}) = 1 - \frac{\sum_{q \in \mathbf{Q}} \mathcal{I}(q, \mathbf{D}_B)}{|\mathbf{Q}|} \tag{6.3}$$

Combined Measure: Using the effectiveness and efficiency definitions in Equations 6.1 and 6.3, we define the combined measure for evaluating the indicator function $\mathcal{I}$ as the weighted harmonic mean of the two:

$$\mathcal{E}_\alpha(\mathcal{I}) = \frac{\text{effect}(\mathcal{I}) \times \text{effic}(\mathcal{I})}{\alpha \times \text{effect}(\mathcal{I}) + (1 - \alpha) \times \text{effic}(\mathcal{I})} \tag{6.4}$$

The weighted harmonic mean amplifies the contributions of the outliers (extremes) in both effectiveness and efficiency. For example, when α is set to 0.5, $\mathcal{E}_{\alpha=0.5}$ does not favor solutions that are highly inefficient even when they are highly effective and vice versa. To achieve a high $\mathcal{E}_{\alpha=0.5}$ a solution must achieve a good effectiveness and efficiency balance.

6.1.3 Learning Threshold-Based Indicator (LTI)

Our goal is to learn an indicator function $\mathcal{I}(q, \mathbf{D}_B)$ that maximizes the combined measure, $\mathcal{E}_\alpha$ (as defined in Equation 6.4). Note that in Equation 6.4, α is a free parameter that can be used to control the relative importance of effectiveness and efficiency. Higher α values favor efficient solutions, whereas lower α values favor effective solutions. α can be chosen depending on the constraints imposed on the meta-search engine such as the cost of querying the candidate rankers or the amount of queries that can be issued to the candidate rankers in a given time period. In this paper, we focus on developing a general solution that can maximize this combined measure $\mathcal{E}_\alpha$, for any given α.

To this end, we build a classifier (LTI) that can be tuned for different α values. Intuitively, the classifier's decision to query the candidate ranker C must depend on α. For large α values, when $\mathcal{E}_\alpha$ favors efficient solutions, the classifier must query C only when the probability of obtaining gains is high. Whereas for small α values, when $\mathcal{E}_\alpha$ favors effective solutions, the classifier can choose to query C, even when the probability of gain is small (but non-zero).

Based on this intuition, we build a binary classifier that predicts for each query, whether the gains from using the candidate ranker will exceed a specified threshold T. Instead of using the classifier's binary decision, we use the classifer's output probability, $P(\mathcal{G}(q) \geq T)$, which represents the probability of the gain measure $\mathcal{G}(q)$ being above a given threshold T. Using this probability, the threshold-based indicator function $\mathcal{I}$ is defined as follows:

$$\mathcal{I}(q, \mathbf{D}_B) = \begin{cases} 1 & \text{if } P(\mathcal{G}(q) > T) \geq p \\ 0 & \text{otherwise} \end{cases} \tag{6.5}$$

If the output probability of the classifier is greater than p, then we query the candidate ranker C. Otherwise, we use the base ranker alone. The performance of the indicator function $\mathcal{I}$ varies for different choices of p. Higher p values reduce the number of times C is queried (i.e., favoring efficient solutions), whereas lower p values increase the number of times C is queried. Therefore, for a given α, we learn p values that maximize $\mathcal{E}_\alpha$.

The performance of the binary classifier which determines the probability $P(\mathcal{G}(q) \geq T)$, directly impacts the performance of the indicator function $\mathcal{I}$. From Equation 6.5 we can see that low classification errors will correspond to high $\mathcal{E}_\alpha$ values. Accordingly, to optimize the performance of the binary classifier we set the threshold T in Equation 6.5 to be the *median* of the gains on a training set of queries. This setting ensures that the number of positive ($\mathcal{G}(q) > T$) and negative ($\mathcal{G}(q) \leq T$)

examples in the training set is balanced, a desired property from a classification standpoint.

6.2 Features

To learn a classifier that predicts the probability $P(\mathcal{G}(q) > T)$, we train a standard logistic regression model

$$P(\mathcal{G}(q) > T) = \frac{1}{1 + e^{\Lambda F_q}},$$

where $F_q = f_{q_1}, \ldots, f_{q_n}$ is a feature vector representing query q and $\Lambda = \lambda_1, \ldots, \lambda_n$ is an associated weight vector, which is optimized to reduce the classification error on a training set.

The features we use for this task must satisfy the constraints of the black-box meta-search scenario. We do not have access to the scores or the internals of the different rankers. As a result, we cannot use the features we developed for ReEff which is based on retrieval scores and retrieval features. Because we do not have access to the internals (such as indexes) of the search engines, we cannot use the traditional *pre-retrieval* query performance predictors such as IDF or PMI (He & Ounis, 2004b; Baeza-Yates et al., 2009) or the *post-retrieval* performance predictors such as Query Clarity (Cronen-Townsend et al., 2002) or Weighted Information Gain (Zhou & Croft, 2007).

Instead, we develop easy-to-compute features that correlate with the expected utility of querying the alternate ranker. We target two types of information. First, we use features that characterize the quality of the baseline ranking. The higher the effectiveness of the baseline ranking, the lower the chances of improving it. Second, we use features that can indicate a high overlap between the baseline ranker and the candidate ranker. Higher overlap in documents also suggests lower chances of obtaining new relevant documents.

Table 6.1: Features used for utility prediction.

Source	Feature Name	Description	Aggregates
q	`qLenTerms`	# terms in the query	
	`qLenChars`	# characters in the query	
	`qAggTermLen`	Term lengths in the query	`Max,Mean`
	`qIsCap`	Does the query contain capitalized terms?	
	`qNStops`	# stopwords in the query	
	`qIsQuestion`	Does the query start with a wh-word?	
	`qWikiNgram`	Fraction of query n-grams appearing as wiki titles	
$\mathbf{D}_B$	`uDepth`	URL depth in $\mathbf{D}_B$	`Max,Mean,Std`
	`uLenChars`	# of characters in URLs in $\mathbf{D}_B$	`Max,Mean,Std`
	`uuOvlp`	Inter-snippet overlap in $\mathbf{D}_B$	`Max,Mean,Std`
	`uEntropy`	Entropy of snippets in $\mathbf{D}_B$	`Max,Mean,Std`
	`uqOvlp`	Query-snippet overlap in $\mathbf{D}_B$	`Max,Mean,Std`
	`uqCover`	Fraction of query terms covered by the snippets in $\mathbf{D}_B$	
	`uqNgramCover`	Fraction of query n-grams covered by the snippets in $\mathbf{D}_B$	
	`uqFullCover`	Does an exact query match appear in one of the snippets in $\mathbf{D}_B$?	

The features we use are shown in Table 6.1. We use the information we can glean from the query itself, such as its length and its grammatical structure (e.g., features `qLenTerms,qLenChars,qIsCap,qIsQuestion`), which were shown to correlate with query performance (Bendersky & Croft, 2009), the structure of URLs (features `uDepth,uLenChars`) and the contents of the snippets in the retrieved list. To estimate *inter-ranker* overlap, we use the *intra-ranker* overlap (overlap between the retrieved snippets - `uuOvlp,uuEntropy`) and *query-ranker* overlap (`uqNgramCover,uqFullCover`) as approximations.

6.3 Experimental Setup

6.3.1 Datasets and Rankers

We conduct selection experiments to evaluate *LTI* for the task of selecting queries for which the candidate ranker C is to be queried. For this task, we use the 150 title queries from the TREC *Gov2* collection, for which relevance judgments are available.

We use the Indri retrieval system (Strohman et al., 2004) to conduct retrieval. We use Query Likelihood (QL) (Ponte & Croft, 1998b) as a baseline ranker, and Okapi BM25 (Robertson et al., 1994) as a candidate ranker. We use QL as the baseline ranker and Okapi BM25 as the candidate ranker as it has a higher retrieval effectiveness (in terms of mean average precision) over QL, and provides significant relevance gains, $\mathcal{G}(q)$, for a larger number of queries.

In keeping with the *black-box* scenario for meta-search, we consider only the top 10 search results from each ranker and compute the gain measure, $\mathcal{G}(q)$ with $K = 10$. To generate features for each query, we only extract the corresponding URL, and the snippet information for the top 10 search results found on the results page. We use the default snippet generation available in Indri, which accumulates text (of fifty words) surrounding the places where query terms match in the retrieved documents.

6.3.2 Baselines

PriorRC - To illustrate the difficulty of the selection problem, we also include a prior-based random classifier approach *PriorRC*. For a given threshold T, *PriorRC* uses the training data to determine the fraction of positive instances

$$fp = \frac{|\{q \in \mathbf{Q} \colon \mathcal{G}(q) > T\}|}{|\mathbf{Q}|}.$$

To assign labels to test set, $\mathbf{T}$, *PriorRC* samples labels from a binomial distribution $Bin(|\mathbf{T}|, fp)$. We report average evaluation measures obtained over 10 different random assignment of labels to the test fold.

Classifier - We also use a standard binary classification solution, *Classifier*, which is trained using the same set of features as the threshold-based classification approach. Similar to the threshold-based approach, *Classifier* first predicts the probability that the gain for a given query exceeds the specified threshold T. However, unlike the thresholded approach, *Classifier* uses a fixed cutoff $p = 0.5$, on the pre-

dicted probability to decide whether to query the candidate ranker (see Equation 6.5). For a given query, *Classifier* includes the candidate ranker only if the output probability for the query exceeds this fixed cutoff 0.5. However, for different values of α in $\mathcal{E}_\alpha$, this fixed setting of p can either be too conservative (if highly effective solutions are preferred) or too lax (if highly efficient solutions are preferred).

In contrast, our threshold-based approach (denoted *LTI*) can learn different values of p for different settings of α. Accordingly, for a given value of α we pick the value of p from the range $[0, 0.05, \ldots, 0.95, 1]$ which yields the highest $\mathcal{E}_\alpha$ on the training queries.

In all the experiments, we report the results obtained using a 3-fold cross-validation.

6.4 Results

In this section we present the results of *LTI* in terms of optimizing for $\mathcal{E}_\alpha$ and an analysis of its performance.

6.4.1 Optimizing $\mathcal{E}_\alpha$

Table 6.2 compares the performance of the three methods in terms of effectiveness and efficiency for three different values of α. *LTI* yields the best performance for all three settings. The *Classifier*'s performance is similar to that of *PriorRC*'s performance, which shows that using the classifier's binary decision alone is not adequate for this task.

Note that both *Classifier* and *PriorRC* do not alter their decisions for different α values and hence always achieve the same effectiveness-efficiency tradeoff. On the other hand, *LTI* adapts to different α settings by learning different thresholds. When α is 0.1, *LTI* increases the number of times the candidate ranker is queried, and thus has higher effectiveness (at the cost of lower efficiency). On the other hand, when α

Table 6.2: Comparison of the fixed binary classifier and the dynamic p setting (LTI). Combined measure $\mathcal{E}_\alpha$, effectiveness and efficiency are reported for different values of α. * indicates statistically significant improvements over *Classifier* using Fisher's randomization test with $p < 0.05$.

Method	α=0.1 (favors effectiveness)			α=0.5			α=0.9 (favors efficiency)		
	$\mathcal{E}_\alpha$	Effect.	Effic.	$\mathcal{E}_\alpha$	Effect.	Effic.	$\mathcal{E}_\alpha$	Effect.	Effic.
PriorRC	0.3870	0.3721	0.6302	0.4634	0.3721	0.6302	0.5862	0.3721	0.6302
Classifier	0.3908	0.3744	0.6443	0.4736	0.3744	0.6443	0.6010	0.3744	0.6443
LTI	0.5642* *(+44%)*	0.5806	0.4497	0.5079* *(+7.2%)*	0.4668	0.5570	0.6229* *(+3.6%)*	0.3294	0.6913

is set to 0.9, *LTI* reduces the number of times the candidate ranker is queried and thus, increases efficiency (at the cost of lower effectiveness).

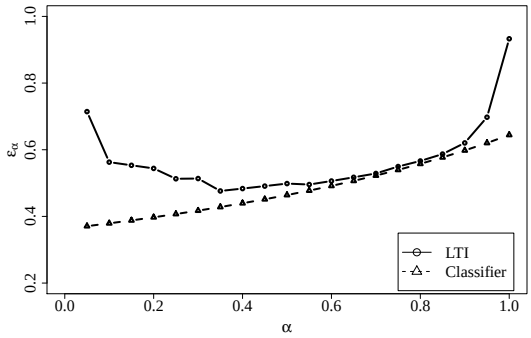


Figure 6.1: Comparison of the dynamic p setting (*LTI*) and the fixed binary classifier for different α values.

Figure 6.1 further illustrates this point. It shows that for most values of α, LTI outperforms the classifier baseline, *Classifier*. For most values of α, the setting of $p = 0.5$ used by the *Classifier* is sub-optimal, in terms of $\mathcal{E}_\alpha$, since it assumes a fixed relative importance between effectiveness and efficiency. This is especially evident for the cases where α is either very large ($\alpha \geq 0.9$) or very small ($\alpha \leq 0.2$) — that is in cases where effectiveness and efficiency are not equally important.

6.4.2 Query Examples

Table 6.3 shows the queries for which *LTI* has the highest and the lowest accuracy in predicting the effectiveness gain. In a query-by-query analysis, we were unable to identify any relationship between a particular type of query and the prediction performance. As can be seen from Table 6.3, the best and the worst performing queries do not vary significantly in either their length or their grammatical structure.

In general, for the best performing cases (the top half of Table 6.3), *LTI* tends to predict correctly the cases in which no major gain is expected from the candidate ranker. These are either the queries with a) the best retrieval performance for which the base ranker already has perfect performance or b) the worst retrieval performance for which both the base ranker and the candidate ranker fail to retrieve relevant documents. On the other hand, the worst performing cases (the bottom half of Table 6.3), are split between extremely high and low relevance gain.

These observations suggest that *LTI* is more reliable at predicting gains when there is higher overlap in the result sets. As we will illustrate later in Section 6.5, when there is low overlap between result sets the likelihood of obtaining relevance gains varies a lot, whereas when there is high overlap the likelihood of gains is much lower and varies less.

6.4.3 Feature Analysis

In this section, we analyze the correlation between the features used to predict the effectiveness gain, and the gain itself. The features that we use to predict the effectiveness gains are described in Table 6.1. In Table 6.4, we show the ten features with the highest value of Spearman's rank correlation coefficient ρ, which demonstrates the rank correlation between these features and the effectiveness gain expected from including the results from the candidate ranker.

Table 6.3: Examples of queries for which the highest and the lowest gain prediction performance is observed.

Query	Predicted	Actual
nuclear reactor types	0	0
david mccullough	0	0
hunting deaths	0	0
urban suburban coyotes	0.09	0.1
low white blood cell count	0.19	0.2
...		
...		
...		
artificial intelligence	0.78	0
history of physicians in america	0.83	0
big dig pork	0.93	0.1
executive privilege	0	0.9
civil air patrol	0	0.9

Note that the most predictive features for estimating effectiveness are based on the result set of the base ranker, rather than the query itself. In fact, the three query dependent features in the Table 6.4 occupy the lowest positions in the table.

The rest of the features in Table 6.4 are based on the results retrieved by the base ranker. Different aggregates of the inter-snippet overlap (`uuOvlp`) within the base result set are inversely proportional to the expected effectiveness gain. That is, a query with highly similar results in the base result set is unlikely to benefit from the results of the candidate ranker. On the other hand, a query that returns potentially relevant, but diverse results is likely to benefit from additional results. This is demonstrated by the positive correlation of `mean` aggregate of the `uqOvlp` features, as well as query coverage features (`uqCover` and `uqNgramCover`), and the negative correlation of the `Std` aggregate of the `uqOvlp` feature.

It is important to note, however, that in all the cases, the values of Spearman's ρ are relatively low, indicating that each feature on its own is not a very reliable predictor of the expected gain. This indicates that our task of learning when to query a

Table 6.4: Ten features with the highest absolute value of Spearman's rank correlation (ρ) with the effectiveness gain.

#	Feature	Aggregate	ρ
1	`uqOvlp`	`Mean`	+0.21
2	`uuOvlp`	`Mean`	−0.20
3	`uqOvlp`	`Std`	−0.20
4	`uuOvlp`	`Std`	−0.16
5	`uqNgramCover`		+0.15
6	`uqCover`		+0.14
7	`uuOvlp`	`Max`	−0.11
8	`qNstops`		+0.09
9	`qAggTermLen`	`Mean`	−0.09
10	`qLenTerms`	`Mean`	+0.09

candidate ranker is challenging, especially given the limited amount of training relevance data. Therefore, in the next section we explore the benefits of using surrogate instances to augment relevance data.

6.5 Learning with Surrogate Gains

LTI's performance depends on the availability of training data in the form of documents manually judged for relevance. Recall that we model the utility of the candidate ranker in terms of its relevance gain, which is defined as the fraction of new *relevant* documents that can be obtained. In practice, this relevance-based training data is usually limited. In this section, we describe a technique to augment this manually judged training data with automatically generated overalp-based data.

6.5.1 Approach

We use *overlap* between the two ranked lists – the number of documents in common, as a surrogate for relevance gain. Given a query q and the two rankings $\mathbf{D}_B$, $\mathbf{D}_C$ we define overlap $\mathcal{O}(q)$, as a fraction of the results in $\mathbf{D}_B$, which appear both in $\mathbf{D}_B$ and in $\mathbf{D}_C$

$$\mathcal{O}(q) = \frac{|\mathbf{D}_B \cap \mathbf{D}_C|}{K}. \tag{6.6}$$

Since computing the *overlap* does not require relevance judgments, we can automatically generate large amounts of this surrogate data.

However, directly utilizing the *overlap* information in place of the relevance based gain is not viable because low overlap between retrieved sets does not always correspond to high relevance gains from their combination (Lee, 1997). To illustrate this point, Figure 6.2 shows the distribution of relevance gains against different levels of overlap between two different retrieval systems for 150 Gov2 TREC topics. As expected, when overlap is high the possible gains are usually low. However, when the overlap is low there is a much higher variance in possible gains.

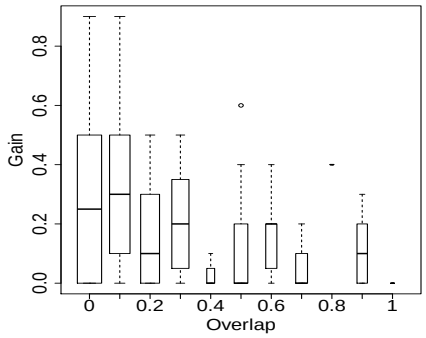


Figure 6.2: Overlap versus Relevance Gains: Distribution of gain values for different levels of overlap in Gov2 queries.

Instead of directly using overlap in place of relevance gain, we first learn a model that learns to map the *overlap* information to the relevance gains. To learn this mapping, we use the data for which relevance judgments are available. Then, for each instance in the automatically generated surrogate data, we obtain the relevance gains predicted using this learned model. This surrogate data, now augmented with predicted relevance gains, is then combined with the original data for training. We formally define this process of learning with *surrogate gain data* as follows.

Let $\mathbf{G}$ denote the set of training queries that have both relevance gain ($\mathcal{G}(q)$) information and overlap ($\mathcal{O}(q)$) information. In addition, let $\mathbf{O}$ denote the set of training queries that only have overlap information but no relevance gain information. Let F_q denote the features associated with each query q (as described in Section 6.2). To obtain the surrogate gains, we first learn a mapping function

$$M : \{F_q, \mathcal{O}(q)\} \rightarrow \mathcal{G}(q),$$

using linear regression on $\mathbf{G}$ that is used as the training data.

Then, we create an augmented training set $\mathbf{G}'$, by applying the mapping function to $\mathbf{O}$ i.e.,

$$\mathbf{G}' = \mathbf{G} \bigcup_{q \in \mathbf{O}} M(F_q, \mathcal{O}(q)). \tag{6.7}$$

Finally, we learn the threshold-based classifier, *LTI*, using this augmented training set. It is important to note that in this process of learning with surrogate gain data, *LTI* is still trained over the same set of features as before and does not use overlap as a feature, since it is not available during testing.

6.5.2 Experiments

We conduct selection experiments to evaluate *LTI* with the augment training data. To create the augmented training set $\mathbf{O}$, we use samples of varying size from the TREC 2008 Million Query Track (*Million* collection), which contains a set of 10,000 title queries. We use the 150 title queries from the Gov2 collection that we used before as the main relevance-based data set $\mathbf{G}$. We have access to both the relevance-based and overlap-based gain information for this collection.

The evaluation involves the following steps:

1. We learn the mapping function $M : \{F_q, \mathcal{O}(q)\} \rightarrow \mathcal{G}(q)$ using the training portion of the Gov2 data set ($\mathbf{G}$).

2. Then, we apply the mapping M to the queries in the *Million* collection, the overlap collection ($\mathbf{O}$), to produce surrogate gains. This surrogate gain data is then combined with the training portion of the Gov2 data set to produce the augmented training data ($\mathbf{G}'$).

3. This augmented training data $\mathbf{G}'$ is then used to train *LTI*, which is the applied to the test set of evaluation.

Note that the test portion of the Gov2 collection is not used for any part of training and the overlap information in the test portion of the Gov2 queries also remains unused.

6.5.3 Results

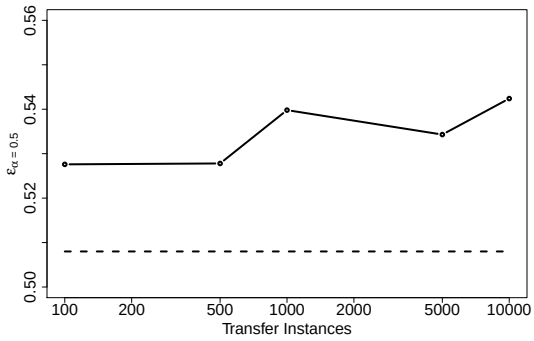


Figure 6.3: Effect of adding increasing amounts of surrogate (transfer) instances on the $\mathcal{E}_\alpha$ measure. The horizontal line corresponds to the performance of *LTI* with no transfer instances from Million.

Adding surrogate instances provides substantial improvements in $\mathcal{E}_{\alpha=0.5}$, ranging from 2% to 8%, over *LTI* trained without any surrogate instances. Figure 6.3 shows the performance of *LTI* in terms of $\mathcal{E}_\alpha$ with $\alpha = 0.5$, when different amounts of surrogate instances are added to the training data. For each subset size, the results are averages over five different subsets of surrogate data of the corresponding size, which were sampled uniformly at random.

Adding large amounts of surrogate instances provides improvements in $\mathcal{E}_\alpha$ for most settings of α. Figure 6.4 shows the $\mathcal{E}_\alpha$ performance of the threshold-based approach when training with all 10000 surrogate instances for different α settings. Overall, adding surrogate instances increases the training data available for both 1) learning the classifier that predicts the probability of the gains exceeding a given threshold, and 2) for learning the cutoff p, on the predicted probabilities. As a result, we find that the classification accuracy – the accuracy of predicting whether the gains exceed a specified threshold – improves substantially, which in turn improves the performance with respect to $\mathcal{E}_\alpha$. For example, for $\alpha = 0.5$, we find that the overall classification accuracy improves by more than 9% when using 10,000 surrogate instances, relative to using no surrogate data.

Most gains from adding surrogate instances are obtained at lower values of α. For α values, $0.1 \leq \alpha \leq 0.6$, adding surrogate instances provides improvements ranging from 8% to 15%. However, for higher values of α, $\alpha \geq 0.6$, adding surrogate instances does not provide substantial improvements.

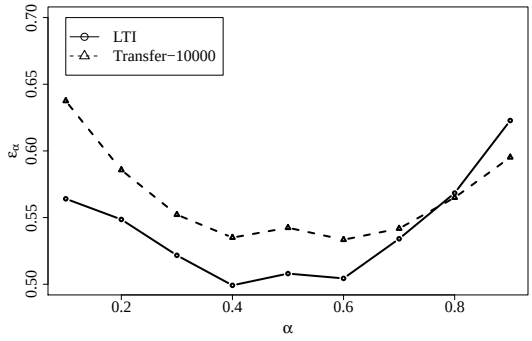


Figure 6.4: Performance of *LTI* with and without surrogate instances for different α settings.

It turns out that adding surrogate instances tends to favor effective solutions, while slightly lowering efficiency. As a result the $\mathcal{E}_\alpha$ improvements are obtained for

lower alpha values, which favor effective solutions. For instance, at $\alpha = 0.1$, we see nearly a 15% improvement. On the other hand, for the higher α settings, which favor efficiency, we do not observe such large improvements in $\mathcal{E}_\alpha$.

An inspection of the classification accuracy shows that the improvements in recall are higher than the improvements in precision. For example, when adding 10,000 surrogate instances, we find that recall increases by 8%, whereas precision only increases by 4%, compared to using no surrogate data. In other words, adding surrogate data leads to an increase in identification of queries whose actual gains exceed the specified threshold, thereby increasing recall. Because surrogate instances are noisy, their addition also leads to false positives and does not provide a corresponding improvement in precision.

Overall, this has an effect of improving effectiveness, while slightly lowering the overall efficiency.

6.6 Summary

In this chapter we explored efficiency constraints when querying alternate rankers in a meta search scenario. While querying alternate rankers can yield improvements in effectiveness, always querying the alternate ranker can be expensive. To model these competing goals, we measure the performance of ranker combination, using $\mathcal{E}_\alpha$ that balances the trade-off between the effectiveness and the efficiency aspects. We develop *LTI*, a threshold-based classiifer that directly optimizes the combined measure, $\mathcal{E}_\alpha$.

Empirical results on a standard web collection demonstrate the utility of *LTI*. Compared to the standard classification method, *LTI* provides more than 7% improvement in the combined measure $\mathcal{E}_{0.5}$, when efficiency and effectiveness are equally important.

A key feature of *LTI* is that it can be adapted to different effectiveness versus efficiency trade-offs, specified via different α settings. The experimental results show *LTI* provides someL improvements for most settings of α, especially in the extremes.

For the cases when the available relevance data is scarce, we develop a technique for automatically generating surrogate training data using the ranker overlap information. Our experimental results show that surrogate data can improve the performance of the *LTI* between 8% to 15%, for settings that favor effectiveness, and does not yield as much benefits when efficiency is favored.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

Query-dependent selection of retrieval alternatives can yield superior performance compared to a fixed choice over all queries. In this thesis, we explored techniques for query-dependent selection of retrieval alternatives. We developed a relative effectiveness estimation technique, ReEff, that leverages features that are already used for retrieval. We showed that this technique can be applied successfully for automatic query reduction and for combining multiple ranking algorithms. We also explored query dependent selection under efficiency constraints in a black-box meta-search scenario. In this chapter, we reiterate the main findings of this thesis, discuss some limitations and lessons learned, and point to some future work.

7.1 Summary of main findings

1. **Estimating Effectiveness of Web Search** - ReEff is an effective technique for modeling effectiveness for web search results but does not perform as well on small scale collections with few features. On large collections with many features, ReEff is more suitable for predicting effectiveness of web search results, than content based measures such as Clarity, as it uses aggregates of features that are already available during retrieval. Experiments on a large scale web collection showed that the aggregates of retrieval features show high correlation with retrieval effectiveness. Experiments on a small scale web collection showed that ReEff is not as effective when trained on small amounts of data with few retrieval features. We also find that ReEff is more reliable for finding hard

queries, especially for queries with $DCG@5 < 0.2$, and is less reliable for finding easy queries.

2. **Applications of ReEff** - Query-dependent selection showed improvements for automatically selecting reduced queries, and for selecting between ranking algorithms. For query reduction the *Difference* formulation yielded an improvement of 4% over using the original long query always. When choosing between five ranking algorithms, *Difference* yielded 3.75% improvement over using the Best-on-Train on baseline. This shows the feasibility of using ReEff for query-dependent applications for web search.

3. **Quality of Impact** - The average performance gains using ReEff tend to be small (less than 5% relative improvement for both applications), as the improvements are obtained for a smaller subset of queries. On a subset of 5% of the queries, we see a nearly 25% relative increase for query-reduction, and a 13% relative increase for choosing between ranking algorithms. A simple thresholding showed that ReEff can be tuned to have a high positive impact ratio of more than 70% on a subset of 3% of the queries. For web search engines such as Google, which handle a large number of queries each day (in the millions), large improvements on five percent of their traffic can have a substantial impact.

4. **Relative versus Independent Estimation** - For both applications, we find that modeling relative effectiveness is better than using independent estimation of performance, when using the retrieval-based features in ReEff. Even though both formulations essentially use the same information, we find that *Difference* outperforms *Independent* for both query-reduction and for selecting between ranking algorithms. Intuitively, focusing on differences between effectiveness is closer to the end goal of selecting the best alternative. The improvements of

Independent when thresholding on the differences further highlights the benefits of modeling relative differences directly.

5. **Potential versus Achieved Impact** - Selection is more useful when there are large gains (large potential). There are two reasons for improvements under large potential. First, when there is large potential, the actual differences in performance are greater than the typical range of errors for effectiveness estimation, which makes selection more likely to be useful. Second, when there is large potential for improvement, the selection problem becomes easier in some respects. For example, when we have a large potential for improvement using reduction, the fraction of reduced queries that are worse than the original tends to be lower, which means the risk of selection is reduced.

6. **Benefitting Poorly Performing Queries** - Queries with poor baseline performance typically have large potential for improvement. Because selection yields some improvements when there is large potential, it benefits poorly performing queries, thereby delivering improvement where it matters the most.

7. **Selection versus Fusion** - Selection and fusion provide different types of benefits. For both query-reduction and ranking algorithms, selection yields large gains over a small subset of the queries, whereas fusion yields smaller gains but over a larger subset.

8. **Selective Fusion** - Further, in the case of ranking algorithms we find that fusion works well for some queries, whereas selection works better for other. We use this observation to perform selective fusion, where we use ReEff to automatically select between the fused result set and the results from the individual rankers. In terms of average performance selective fusion yields small additional improvements over both selection (1%) and fusion (2%). More importantly, se-

lective fusion also improves the robustness with respect to the Best-on-Train baseline nearly doubling the robustness index (RI) of selection and fusion.

9. **Ranker Selection and Weighting for Fusion** - Experiments showed that ReEff provides used for query-dependent selection of a subset of rankers to improve fusion using three state-of-the-art fusion approaches. The query-dependent weights assigned by ReEff also yields small improvements query-independent weights used for fusion.

10. **Efficiency** - In a black-box meta-search scenario, we investigated query-dependent selection under a specific efficiency constraint, where querying all available rankers for all queries is expensive. Querying alternate rankers can lead to increased effectiveness gains but can be inefficient. To model these effectiveness and efficiency goals we developed a combined measure that can be parametrized to characterize different trade-offs. Using easy to compute features based on the results set of a single ranker, we developed a classifier that predicts when querying an alternate ranker is useful. The classifier did not perform much better than a simple prior-based random selection.

    However, by learning thresholds on the classifier's probabilities, we showed that the performance of selection can be improved by more than 7% in terms of $\mathcal{E}_{0.5}$, the combined measure that we developed. Further, we showed that this thresholded-approach adapts well for most settings of α that characterize the different trade-offs in effectiveness and efficiency.

11. **Leveraging Surrogate Data** - We showed that automatically generated overlap information can be used to augment the relevance-based training data. This approach yields substantial improvements over using the limited relevance-based data alone, except in settings where efficiency is important (for settings of $\alpha > 0.6$).

7.2 Why is relative effectiveness estimation useful?

We find that using simple aggregates of the ranking scores and retrieval based features can yield improvements for query-dependent selection. In this section, we hypothesize some mechanisms based on our experimentation to explain why ReEff can be useful.

7.2.1 Features

A main reason for ReEff's performance is because it is able to identify poor performance reliably. We find that most gains that ReEff achieves are cases where the baseline performance is poor and the risk involved in selection is lower. This suggests that some part of ReEff's performance is because its features are more reliable at identifying poor performance.

We hypothesize that the features used in ReEff are crude measures of certain aspects of result sets that correlate with their effectiveness. For example, the variance of the scores (relevance estimates) of the top-ranked documents can be viewed as an approximation for the robustness of the top ranked documents, which has been shown to correlate well with retrieval. The mean of the top-ranked documents is an un-normalized aggregation of individual relevance estimates, which can be viewed an approximation to the weighted information gain measure (Zhou & Croft, 2007).

We hypothesize that these crude approximations can yield strong prediction performance, when learnt from large amounts of training data and when these approximations are computed over multiple estimates of relevance. The performance characteristics of ReEff under different amount of training data and the poor performance of ReEff on small scale collections (shown in Section 7.3.2) lend some evidence to this hypothesis.

7.2.2 Formulation

The application of ReEff to two different selection problems shows that targeting relative improvements with respect to a baseline is better than modeling independent estimation of individual effectiveness. We hypothesize the main causes to be a) the disconnect between independent estimation objective and the selection objective, and b) the query-level variance in feature values.

When selecting between multiple alternatives, we want selection to be better than the simple best-on-train baseline. Therefore, the selection objective is to select an alternative that is better than this designated baseline. On the other hand, the independent estimation's objective is to minimize prediction errors of absolute effectiveness of each alternative. As a result, it is possible for independent estimation to learn a model that minimizes the difference with absolute effectiveness well, while still performing poorly when ranking the alternatives with respect to the baseline. This is analogous to the benefits seen for learning-to-rank methods that train on pair-wise preferences between documents (Joachims, 2002), when compared to point-wise methods such as regression that train on individual document relevance.

The query-level variance in ranking score and retrieval features can impact the learning. For example, the ranking scores while useful for discriminating between individual documents retrieved for the same query are not comparable across queries. Using relative estimation forces the model to normalize the scores and feature values relative to the baseline, which can make the values more comparable across different queries.

7.3 Limitations

7.3.1 Potential versus Actual Performance

ReEff's selection performance is far from optimal. For example, ReEff yields only 9% of the gains that an oracle can achieve for selecting between ranking algorithms.

A main contributing factor is the inability of ReEff to reliably model small differences in effectiveness. We find that the typical range of prediction errors for ReEff is high, up to 0.4 in some cases, which makes it harder to detect small differences (less than 0.1 in MAP). As a consequence, ReEff is unable to exploit a large proportion of the potential for selection. As shown in Figure 7.1(a) nearly 80% of the possible selection gains are small (less than 0.1 in MAP).

We believe that ReEff's difficulty with modeling small differences is related to a higher overlap in results sets. Small differences in result sets appear to correlate with high overlap in result sets. As shown in Figure 7.1(b) when the differences in average precision are smaller the overlap between result sets tends to be higher.

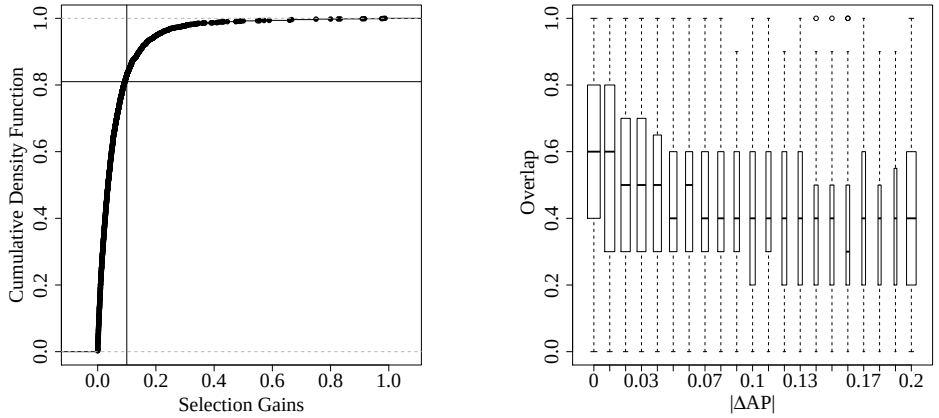
The result sets that have a high overlap can differ in their ranking effectiveness due to their differences in the ranking of the overlapping documents. However, most of the retrieval-based features used in ReEff are aggregate features of the top ranked documents that are insensitive to different orderings of the same documents. As a result ReEff cannot reliably model small differences in effectiveness between result sets that have a high overlap.

7.3.2 Large Training Data

In this work, we demonstrated the utility of query-dependent selection when there are large amounts of training data and retrieval features that are used for Web search.

We compare *Difference* with Best-on-Train for varying training set sizes. Best-on-Train selects the ranker that performs the best on the training set of queries and uses the same ranker for all test queries. *Difference* uses the Best-on-Train ranker as its baseline ranker to perform query-dependent selection.

Figure 7.2 compares the performance of *Difference* and Best-on-Train when we vary the amount of training data used. For each training set size that is less than



(a) CDF of gains possible through selection.

(b) Fraction of overlap between the result sets plotted against the absolute difference in Average Precision ($|\Delta AP|$).

Figure 7.1: Distribution of selection gains and relationship to overlap between result sets. In (a) we only show the distribution of positive differences (gains). In (b) we collapse the differences that are greater than 0.1 into a single bin.

2000 instances, we run Best-on-Train and *Difference* on five different random samples to account for variance due to sample selection.

The plot shows that increasing amounts of training data improves the performance of both Best-on-Train and *Difference* as expected. For smaller amounts of training data (less than 2000 instances), Best-on-Train yields better generalization i.e., selecting a single ranker that does the best on training data is better than a query-dependent selection using ReEff when only small amounts of training data are available. ReEff trains a non-linear regression model over a large feature space, more than 500 features in the case of ranking algorithms, which in part explains the need for large amounts of training data.

Recall that our evaluation of ReEff for effectiveness estimation on the smaller Gov2 collection (Section 3.2.3) also showed that ReEff was not as effective when training on smaller data set using small number of features.

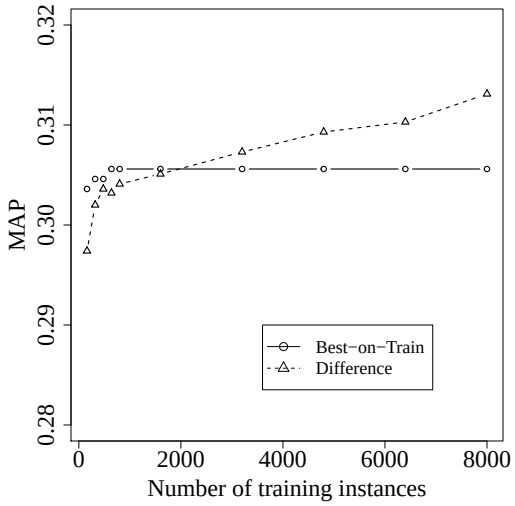


Figure 7.2: Learning curve for selection: MAP Performance of the *Difference* formulation for varying amounts of training data. For training sizes less than 2000, we use averages obtained over 5 runs of each size to account of variance due to sample selection.

We also evaluated ReEff for query reduction on a smaller collection. The query reduction experiment on 150 queries from the Gov2 collection was designed to evaluate the feasibility of query-reduction when using a small number of features. The results showed modest (but not significant) gains (see Table 4.6 in Chapter 4).

We also conducted experiments to select between two retrieval models that do not use a large number of features like ranking algorithms used for web search. Table 7.1 shows the performance of selecting between BM25 (Robertson et al., 1994) and an expansion-based retrieval model, RM (Lavrenko & Croft, 2001). We used the aggregates of retrieval scores as features for ReEff. We find the gains due to selection are not significant, less than 1% relative improvement over the expansion model RM, which is used as the baseline. The subset gains are higher but, as with the larger collection, the impact is on less than 10% of the queries.

These results show that ReEff only yields small improvements on small scale collections with fewer retrieval features.

Table 7.1: Query-dependent selection on 250 TREC Robust queries for two retrieval models: BM25 - Alternative, and RM - Baseline.

Baseline	Alternative	Selection	Alt. Queries	Positive	Negative	Subset Gains
0.2805	0.2406	0.2825	21	13	8	0.0203

7.3.3 Efficiency

Compared to other post-retrieval approaches such as Clarity (Cronen-Townsend et al., 2002) which rely on analyzing the content of retrieved documents, ReEff is more efficient as it uses features that are readily available during retrieval. This makes it more suitable for query-dependent applications in web search. However, query-dependent selection can become infeasible when there are large number of alternatives to evaluate. The main bottleneck in evaluating a large number of alternatives is the need to run multiple searches for a single query.

When using multiple ranking algorithms, this problem can be somewhat alleviated by a two stage process, where a smaller subset of documents is first extracted by using a baseline ranker. Then, the other ranking algorithms are used to re-rank this smaller subset.

The analogous solution for query reductions or for other forms of generating query representations can be inadequate. Modifications in query representations lead to substantially diverse results which may not be contained in the results retrieved for the original query representation alone. We used a naive approximation that evaluated reduced queries that are obtained by dropping one word at a time and ignoring the other possible reductions. Considering more reductions increases the potential but can be prohibitively expensive.

Moreover, even running a fixed (smaller) number of additional alternatives during ranking can be expensive for search engines that handle large query volumes. In Chapter 6, we explored a specific variant of this problem under the meta-search

scenario where we designed a classifier that can predict when querying an alternate ranker can be useful. A similar solution can be applied to address the efficiency concerns. We can extend ReEff as a two-staged approach that first stage uses a baseline ranker to decide whether evaluating the alternate rankers is likely to be useful. If found to be useful, the alternatives are then evaluated using ReEff to select the best alternative for the given query.

7.4 Lessons Learned

In this thesis, we investigated query-dependent selection approach for two specific applications namely automatic query reduction and combining ranking algorithms. In this section, we summarize some lessons learned through this investigation, which can benefit future applications of query-dependent selection.

1. It is useful to analyze the the distribution of prediction errors of ReEff, when using it for applications that rely on effectiveness estimation. Our analysis showed that the high values of average prediction effectiveness masks the variance in the performance of ReEff for queries with different effectiveness levels. In particular, we find that ReEff is more reliable for detecting poor performance (Section 3.2.3), which suggests that it can be more useful for applications such as interactive query reformulations rather than for applications such as selective query expansion that rely more on detecting good performance accurately.

2. Analyzing the distribution of the gains that are possible through query-dependent selection can shed light on the suitability of ReEff for the given set of alternatives.

   ReEff is able to detect large differences in effectiveness more reliably and is insensitive to small differences. We saw this trend for both query-reduction (Section 4.4.2), as well as for ranking algorithms (Section 5.3.2). If most of the

potential gains through the alternatives are small, then we are unlikely to see benefits for a large number of queries.

ReEff may not yield substantial gains if there is a single dominant baseline. In such cases, there will be few training instances where the alternatives are better than the baseline. As a result, the trained regression models can be overly conservative and may always choose the baseline. Thus, it is important to know whether the potential gains of the available alternatives is not limited to a few instances.

A small number of alternatives can yield benefits through query-dependent selection, as long as there is a large potential for improvement. As our experiments with the ranking algorithms showed, nearly 80% of the gains can be achieved by choosing between two algorithms (Section 5.3.4).

3. Detecting differences between result sets generated for the same query is a hard problem, especially when there is high overlap between the results. ReEff performs well when there are large amounts of training data. With small training data, a simpler query-independent best-on-train ranker selection performs better than query-dependent ranker selection using ReEff (Section 7.3.2).

   Moreover, ReEff's performance also depends on the availability of several result set based features for estimating effectiveness. Our experiments on small scale collections with fewer features did not show consistent gains through selection (Section 7.3.2).

4. Selection is a risky process that provides large improvements for some queries but also leads to poor performance in some. A simple thresholding on the predicted differences between alternatives improves the robustness of selection (Sections 4.4.3 and 5.3.3).

5. Selection and fusion yield different types of benefits which can be combined to leverage the strengths of both techniques. When fusing all available alternatives is ineffective, a query-dependent selection of alternatives can be helpful. As shown in Section 5.2.3, selection can provide improvements over fusing all rankers, and selective fusion – fusion for some queries and selection for other – can provide improvements over selection and fusion.

6. Predicting the relevance-based gains from an alternate ranker is hard. The binary classifier using the features that we developed based on the results page alone did not perform better than a simple prior-based random model. However, we find that a) learning thresholds on the classifier's probabilities (Section 6.4) and b) utilizing large amounts of training data, even if noisy, can help improve performance for this task (Section 6.5.3).

7.5 Future Work

The availability of large scale data for Web search makes it feasible to investigate query-dependent strategies for retrieval. In this thesis, we considered automatic query reduction and query-dependent selection of ranking algorithms. Natural extensions to this work include other applications of query-dependent strategies, improving the robustness and efficiency of selection and moving into context-dependent strategies.

7.5.1 Other Applications

Search engines often provide related and suggested queries to help users better formulate their needs. When providing suggested queries, a key challenge is to ensure that the suggested queries are of high-quality. This is often done by examining user behavior on past sessions using query logs (Baeza-Yates et al., 2005; Cao et al., 2008; Song & He, 2010), which limits the application to queries that have adequate support. In order to extend this to tail queries, those with little support, we need to

be able to effectively estimate the performance of the suggestions in the absence of click information. ReEff can be useful for ranking suggestions as well as for avoiding poor suggestions for queries when there are no effective suggestions that are available. Also, for long queries the support problem can also be alleviated to by using query reduction techniques to reduce the original query to versions that have more support in the query logs.

7.5.2 Robustness and Efficiency.

The selection performance has a large scope for improvement. In particular we find that quality of impact is high only when affecting a small subset of queries. This robustness of selection performance – i.e., the ratio of positively and negative impacted queries, improves with simple thresholding and fusion of results. This suggests developing a more principled approach that can explicitly model the risk and reward trade-offs involved in selection can further improve robustness. Collins-Thompson (2009) develops a convex optimization framework that models risk-vs-reward for robust query-expansion. In a similar fashion, we can select a seed set of alternatives that guarantee robustness, while also retaining a large fraction of the potential for query-dependent selection.

Evaluating alternatives during retrieval can be expensive due to the computations involved in feature generation, the sheer number of alternatives to consider or constraints on the availability of the alternatives. Therefore, an understanding of the effectiveness versus efficiency trade-offs involved in selection is essential from a practical standpoint. A two-staged approach that first reduces the number of alternatives to consider using more efficient (but less effective) features, followed by a more expensive but more effective analysis using ReEffcan help improve efficiency.

7.5.3 Context-dependent Strategies

Session history, the sequence of queries issued by a user and her actions in response to the presented results, is a valuable resource for modeling user context and user intent. Session-based context models can be used to build better query models for ranking (White et al., 2010), help improve query suggestions (Cao et al., 2008), as well as help to predict switching to alternate search engines (White & Dumais, 2009). One of the key challenges in building session based context models is that there can be large variance in aggregate user behavior depending on query types and on the effectiveness of the retrieved results themselves. For example, for navigational queries users are likely to click on fewer results compared to informational queries. Therefore, session context models can be beneficial for some queries and harmful for others. ReEff's features can be useful for predicting the effectiveness of the session-context model for each query and to develop a technique for selective application of session-based context. Further, the estimated relative effectiveness of search results can be used to further improve session context models by learning a query-dependent mixing weight for the context models.

CREDITS

I thank Giridhar Kumaran and Vitor Carvalho for their advice and guidance for the work described in Chapters 3 and 4. Their guidance and advice was especially useful in focusing my ideas and designing my experiments towards the goal of obtaining efficient features for predicting web search and using them for evaluating them for query reduction.

I thank Michael Bendersky for his help with the work done in Chapter 6. Our discussions led to the focus on the *black-box* meta-search scenario, where there can be cost constraints on querying rankers. I very much value his useful inputs on feature design and useful discussion on formalizing the evaluation measures that we used.

I also thank Van Dang for providing me with RankLib (`http://www.cs.umass.edu/~vdang/ranklib.html`), a learning to rank library that was used for experiments in Chapter 5.

APPENDIX: EXPERIMENTAL COLLECTIONS

The main types of retrieval features used in the large web collections used in this thesis are listed below. The listing is extracted from the web page `http://research.microsoft.com/en-us/projects/mslr/feature.aspx`, which also contains additional information about the features. This is a complete listing of the features in the publicly available learning-to-rank collection used in Chapter 5 ; more information about this collection is available at `http://research.microsoft.com/en-us/projects/mslr/default.aspx`. The experiments in Chapters 3 and 4 use a subset of these features and includes additional variants of the retrieval features and click-based features, which are not disclosed for proprietary reasons.

There are two sets of retrieval features – *query-document* based features and *document* based features.

Query-Document Features

The query-document features listed below are computed over different fields (or streams) in the web page namely, title, URL, anchor text, body, and whole page. The click features are computed for the whole page alone.

- **Coverage** - Number of terms that are covered in the field, and the fraction of query terms covered in the field.
- **IDF** - The aggregates of inverse document frequency of the query terms.
- **TF** - Sum, Min, Max, Mean and Variance of term frequencies of the query terms.

- **Normalized TF** - Sum, Min, Max, Mean, and Variane of field length normalized term frequencies.
- **TF-IDF** - Min, Max, Mean, and Variance of tf*idf.
- **Retrieval Scores** - Boolean model, Vector space model, and BM25.
- **Retrieval Scores** - Language modeling score (LM) with absolute discounting smoothing, with Dirichlet smoothing, and with Jelinek Mercer smoothing.
- **Click Features** - Click count of a query-URL pair at a search engine in a period and its normalized variants.

Document-based Features

The following are the list of document-based features.

- **Stream length** - Length of different fields (title, anchor, body etc.).
- **URL features** - Number of slashes in URL, and length of URL.
- **Anchor features** - Number of inlinks and outlinks.
- **Document Quality** - PageRank, Site level page rank, and web page quality classifier scores (goodness and badness scores).
- **Click features** -The click count of a URL aggregated from user browsing data in a period and the average dwell time of a URL aggregated from user browsing data in a period.

BIBLIOGRAPHY

Agichtein, E., Brill, E., & Dumais, S. (2006). Improving web search ranking by incorporating user behavior information. In *Proceedings of ACM SIGIR conference on Research and development in information retrieval*, (pp. 19–26).

Amati, G., Carpineto, C., Romano, G., & Bordoni, F. U. (2004). Query difficulty, robustness, and selective application of query expansion. In *Proceedings of European Conference on Information Retrieval*, (pp. 127–137).

Arguello, J., Callan, J., & Diaz, F. (2009). Classification-based resource selection. In *Proceedings of Conference on Information and knowledge management*, (pp. 1277–1286).

Aslam, J. A. & Montague, M. (2001). Models for metasearch. In *Proceedings of ACM SIGIR conference on Research and development in information retrieval*, (pp. 276–284).

Aslam, J. A., Pavlu, V., & Yilmaz, E. (2005). Measure-based metasearch. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 571–572)., New York, NY, USA. ACM.

Attar, R. & Fraenkel, A. S. (1977). Local feedback in full-text retrieval systems. *Journal of the ACM (JACM)*, *24*(3), 397–417.

Baeza-Yates, R., Hurtado, C., & Mendoza, M. (2005). Query recommendation using query logs in search engines. In *Current Trends in Database Technology-EDBT 2004 Workshops*.

Baeza-Yates, R., Murdock, V., & Hauff, C. (2009). Efficiency trade-offs in two-tier web search systems. In *Proceedings of the ACM SIGIR conference on Research and development in information retrieval*, (pp. 163–170).

Balasubramanian, N. & Allan, J. (2010). Learning to select rankers. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, (pp. 855–856)., New York, NY, USA. ACM.

Balasubramanian, N., Bendersky, M., & Allan, J. (2010). Cost-effective combination of multiple rankers: Learning when not to query. In *Proceedings of the NESCAI Student Colloquium.*, NESCAI '10.

Balasubramanian, N., Kumaran, G., & Carvalho, V. R. (2010a). Exploring reductions for long web queries. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, (pp. 571–578)., New York, NY, USA. ACM.

Balasubramanian, N., Kumaran, G., & Carvalho, V. R. (2010b). Predicting query performance on the web. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, (pp. 785–786)., New York, NY, USA. ACM.

Bartell, B. T., Cottrell, G. W., & Belew, R. K. (1994). Automatic combination of multiple ranked retrieval systems. In *SIGIR '94: Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 173–181)., New York, NY, USA. Springer-Verlag New York, Inc.

Beitzel, S., Frieder, O., Jensen, E., Grossman, D., Chowdhury, A., & Goharian, N. (2003). Disproving the fusion hypothesis: An analysis of data fusion via effective information retrieval strategies. In *Proc. of SAC*, (pp. 827–832). ACM.

Beitzel, S., Jensen, E., Chowdhury, A., Grossman, D., Goharian, N., & Frieder, O. (2004). Recent Results on Fusion of Effective Retrieval Strategies in the Same Information Retrieval System. In *Lecture notes in computer science Vol. 2924* (pp. 101–111). Springer.

Belkin, N. J., Cool, C., Croft, W. B., & Callan, J. P. (1993). The effect multiple query representations on information retrieval system performance. In *Proceedings of the 16th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '93, (pp. 339–346)., New York, NY, USA. ACM.

Bendersky, M. & Croft, W. B. (2009). Analysis of long queries in a large scale search log. In *WSCD '09: Proceedings of the 2009 workshop on Web Search Click Data*, (pp. 8–14)., New York, NY, USA. ACM.

Bian, J., Liu, T.-Y., Qin, T., & Zha, H. (2010). Ranking with query-dependent loss for web search. In *WSDM '10: Proceedings of the third ACM international conference on Web search and data mining*, (pp. 141–150)., New York, NY, USA. ACM.

Buckley, C., Salton, G., Allan, J., & Singhal, A. (1995). Automatic query expansion using smart: Trec 3. In *Overview of the Third Text REtrieval Conference (TREC-3)*, (pp. 69–80).

Burges, C., Ragno, R., & Le, Q. (2007). Learning to rank with nonsmooth cost functions. *Advances in Neural Information Processing Systems*, *19*, 193.

Cao, H., Jiang, D., Pei, J., He, Q., Liao, Z., Chen, E., & Li, H. (2008). Context-aware query suggestion by mining click-through and session data. In *SIGKDD*.

Carterette, B. & Jones, R. (2008). Evaluating search engines by modeling the relationship between relevance and clicks. *Advances in Neural Information Processing Systems*, *20*, 217–224.

Cetintas, S. & Si, L. (2007). Exploration of the tradeoff between effectiveness and efficiency for results merging in federated search. In *Proceedings of the ACM SIGIR conference on Research and development in information retrieval*, (pp. 707–708).

Chen, Y. & Zhang, Y.-Q. (2009). A query substitution - search result refinement approach for long query web searches. In *Web Intelligence and Intelligent Agent Technology*, (pp. 245–251)., Washington, DC, USA. IEEE Computer Society.

Collins-Thompson, K. (2009). Reducing the risk of query expansion via robust constrained optimization. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, (pp. 837–846)., New York, NY, USA. ACM.

Collins-Thompson, K. & Callan, J. (2007). Estimation and use of uncertainty in pseudo-relevance feedback. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 303–310)., New York, NY, USA. ACM.

Cormack, G., Clarke, C., & Buettcher, S. (2009). Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, (pp. 758–759). ACM.

Croft, W. B. (1981). Incorporating different search models into one document retrieval system. *SIGIR Forum*, *16*(1), 40–45.

Croft, W. B. & Harper, D. J. (1979). Using probabilistic models of document retrieval without relevance information. *35*(4), 285–295.

Croft, W. B. & Thompson, R. H. (1984). The use of adaptive mechanisms for selection of search strategies in document retrieval systems. In *Proc. of the third joint BCS and ACM symposium on Research and development in information retrieval*, (pp. 95–110)., New York, NY, USA. Cambridge University Press.

Cronen-Townsend, S., Zhou, Y., & Croft, W. (2004). A framework for selective query expansion. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, (pp. 236–237). ACM New York, NY, USA.

Cronen-Townsend, S., Zhou, Y., & Croft, W. B. (2002). Predicting query performance. In *Proc. of SIGIR*, (pp. 299–306).

Dogpile.com (2007). Different engines, different results: Web searchers not always finding what theyre looking for online. `http://www.infospaceinc.com/files/Overlap-DifferentEnginesDifferentResults.pdf`.

Efthimiadis, E. N. & Biron, P. V. (1993). Ucla-okapi at trec-2: Query expansion experiments. In *Text REtrieval Conference*, (pp. 278–290).

Farah, M. & Vanderpooten, D. (2007). An outranking approach for rank aggregation in information retrieval. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 591–598)., New York, NY, USA. ACM.

Fox, E. & Shaw, J. (1994). Combination of multiple searches. In *Proceedings of the third Text REtrieval Conference (TREC-3)*, (pp. 243–252).

Freund, Y., Iyer, R., Schapire, R. E., & Singer, Y. (2003). An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research*, *4*, 933–969.

Geng, X., Liu, T.-Y., Qin, T., Arnold, A., Li, H., & Shum, H.-Y. (2008). Query dependent ranking using k-nearest neighbor. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 115–122)., New York, NY, USA. ACM.

Hauff, C., Hiemstra, D., & de Jong, F. (2008). A survey of pre-retrieval query performance predictors. In *CIKM '08: Proceedings of the 17th ACM Conference on Information and Knowledge Management*, (pp. 1419–1420).

Hauff, C., Murdock, V., & Baeza-Yates, R. (2008). Improved query difficulty prediction for the web. In *CIKM '08: Proceedings of the 17th ACM conference on Information and Knowledge Management*, (pp. 439–448).

He, B. & Ounis, I. (2004a). A query-based pre-retrieval model selection approach to information retrieval. In *Proceedings of RIAO*, volume 2004.

He, B. & Ounis, I. (2004b). Inferring query performance using pre-retrieval predictors. In *The Eleventh Symposium on String Processing and Information Retrieval*, (pp. 43–54). LNCS, Springer.

He, J., Larson, M., & de Rijke, M. (2008). Using coherence-based measures to predict query difficulty. In *ECIR '08: Proceedings of the 30th European Conference on Information Retrieval*, (pp. 689–694). Springer, Springer.

Järvelin, K. & Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems*, *20*(4), 422–446.

Joachims, T. (2002). Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 133–142)., New York, NY, USA. ACM.

Klementiev, A., Roth, D., Small, K., & Titov, I. (2009). Unsupervised rank aggregation with domain-specific expertise. In *IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence*, (pp. 1101–1106)., San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Kumaran, G. & Allan, J. (2007). A case for shorter queries, and helping users create them. In *HLT/NAACL*.

Kumaran, G. & Carvalho, V. (2009). Reducing long queries using query quality predictors. In *Proceedings of the ACM SIGIR conference on Research and development in information retrieval*.

Larkey, L. S. & Croft, W. B. (1996). Combining classifiers in text categorization. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 289–297)., New York, NY, USA. ACM.

Lavrenko, V. & Croft, W. B. (2001). Relevance based language models. In *SIGIR '01: 24th Annual ACM SIGIR Conference Proceedings*, (pp. 120–127).

Lee, C., Lin, Y., Chen, R., & Cheng, P. (2009). Selecting Effective Terms for Query Formulation. In *Proceedings of the Fifth Asia Information Retrieval Symposium*, (pp. 168). Springer.

Lee, C.-J., Chen, R.-C., Kao, S.-H., & Cheng, P.-J. (2009). A term dependency-based approach for query terms ranking. In *Proceedings of the Conference on Information and Knowledge Management*, (pp. 1267–1276)., New York, NY, USA. ACM.

Lee, J. H. (1995). Combining multiple evidence from different properties of weighting schemes. In *SIGIR '95: Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 180–188)., New York, NY, USA. ACM.

Lee, J. H. (1997). Analyses of multiple evidence combination. In *SIGIR '97: Proceedings of the 20th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 267–276)., New York, NY, USA. ACM.

Lee, K. S., Croft, W. B., & Allan, J. (2008). A cluster-based resampling method for pseudo-relevance feedback. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 235–242)., New York, NY, USA. ACM.

Liaw, A. & Wiener, M. (2002). Classification and regression by randomforest. *R News*, *2*(3), 18–22.

Lillis, D., Toolan, F., Collier, R., & Dunnion, J. (2006). Probfuse: a probabilistic approach to data fusion. In *Proceedings of ACM SIGIR conference on Research and development in information retrieval*, (pp. 139–146).

Lillis, D., Zhang, L., Toolan, F., Collier, R. W., Leonard, D., & Dunnion, J. (2010). Estimating probabilities for effective data fusion. In *Proceedings of ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, (pp. 347–354).

Liu, T., Xu, J., Qin, T., Xiong, W., & Li, H. (2007). Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proceedings of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, (pp. 3–10). Citeseer.

Manmatha, R., Rath, T., & Feng, F. (2001). Modeling score distributions for combining the outputs of search engines. In *Proceedings of SIGIR conference on Research and development in information retrieval*.

Metzler, D. (2007). Automatic feature selection in the markov random field model for information retrieval. In *Proceedings of Conference on Information and Knowledge Management*.

Metzler, D. & Croft, W. B. (2005). A markov random field model for term dependencies. In *Proc. of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 472–479).

Montague, M. & Aslam, J. A. (2002). Condorcet fusion for improved retrieval. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, (pp. 538–548)., New York, NY, USA. ACM.

Musser, J. (2007). 12 ways to limit an API. Programmable Web Blog, `http://blog.programmableweb.com/2007/04/02/12-ways-to-limit-an-api/`.

Peng, J., Macdonald, C., & Ounis, I. (2009). Learning to select a ranking function.

Piatko, C., Mayfield, J., McNamee, P., & Cost, S. (2004). JHU/APL at TREC 2004: Robust and terabyte tracks. In *In Proceedings of Text REtrieval Conference*.

Plachouras, V., Ounis, I., & Cacheda, F. (2004). Selective combination of evidence for topic distillation using document. In *Proceedings of RIAO*, (pp. 610–622).

Ponte, J. & Croft, W. (1998a). A language modeling approach to information retrieval. In *Proceedings of ACM SIGIR conference on Research and development in information retrieval*, (pp. 275–281).

Ponte, J. & Croft, W. B. (1998b). A language modeling approach to information retrieval. In *Proceedings of the ACM SIGIR conference on Research and development in information retrieval*, (pp. 275–281).

Qin, T., Zhang, X.-D., Wang, D.-S., Liu, T.-Y., Lai, W., & Li, H. (2007). Ranking with multiple hyperplanes. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 279–286)., New York, NY, USA. ACM.

Robertson, S. (1977). The Probability Ranking Principle in IR. *Journal of Documentation*, *33*, 294–304.

Robertson, S., Walker, S., Jones, S., Hancock-Beaulieu, M. M., & Gatford, M. (1994). Okapi at TREC-3. In *Proc. of TREC-3*, (pp. 109–126).

Selberg, E. & Etzioni, O. (1995). Multi-service search and comparison using the MetaCrawler. In *Proceedings of International conference on the World Wide Web*, (pp. 169–173).

Si, L. & Callan, J. (2005). Modeling search engine effectiveness for federated search. In *Proceedings of the ACM SIGIR conference on Research and development in information retrieval*, (pp. 83–90).

Song, Y. & He, L.-w. (2010). Optimal rare query suggestion with implicit user feedback. In *International World Wide Web Conference*, (pp. 901–910).

Spärck-Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*, 11–21.

Strohman, T., Metzler, D., Turtle, H., & Croft, W. B. (2004). Indri: A language model-based search engine for complex queries. In *Proceedings of International Conference on Intelligence Analysis*.

Tao, T. & Zhai, C. (2006). Regularized estimation of mixture models for robust pseudo-relevance feedback. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 162–169)., New York, NY, USA. ACM.

Tomlinson, S. (2004). Robust, Web and Terabyte Retrieval with Hummingbird SearchServer TM at TREC 2004. In *Proceedings of TREC 2004*.

Turtle, H. & Croft, W. B. (1991). Evaluation of an inference network-based retrieval model. *ACM Transactions on Information Systems*, *9*, 187–222.

White, R., Bennett, P., & Dumais, S. (2010). Predicting short-term interests using activity-based search context. In *CIKM*.

White, R. W. & Dumais, S. T. (2009). Characterizing and predicting search engine switching behavior. In *CIKM*.

White, R. W., Richardson, M., Bilenko, M., & Heath, A. P. (2008). Enhancing web search by promoting multiple search engine use. In *Proceedings of the ACM SIGIR conference on Research and development in information retrieval*, (pp. 43–50).

Xu, J. & Croft, W. B. (1996). Query expansion using local and global document analysis. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 4–11)., New York, NY, USA. ACM.

Xu, J. & Li, H. (2007). Adarank: a boosting algorithm for information retrieval. In *Proceedings of ACM SIGIR conference on Research and development in information retrieval*, (pp. 398).

Zhao, Y., Scholer, F., & Tsegay, Y. (2008). Effective pre-retrieval query performance prediction using similarity and variability evidence. In *ECIR '08: Proceedings of the 30th European Conference on Information Retrieval*, (pp. 52–64). Springer.

Zhou, Y. & Croft, W. B. (2006). Ranking robustness: a novel framework to predict query performance. In *CIKM '06: Proceedings of the 15th ACM international conference on Information and Knowledge Management*, (pp. 567–574).

Zhou, Y. & Croft, W. B. (2007). Query performance prediction in web search environments. In *SIGIR '07: 30th Annual ACM SIGIR Conference Proceedings*, (pp. 543–550).

Zhu, Z. A., Chen, W., Wan, T., Zhu, C., Wang, G., & Chen, Z. (2009). To divide and conquer search ranking by learning query difficulty. In *CIKM '09: Proceeding of the 18th ACM conference on Information and knowledge management*, (pp. 1883–1886)., New York, NY, USA. ACM.

Zighelnic, L. & Kurland, O. (2008). Query-drift prevention for robust query expansion. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, (pp. 825–826)., New York, NY, USA. ACM.