Learning To Reformulate Long Queries

by

Neha Gupta

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2010

Author .....
Department of Electrical Engineering and Computer Science
May 21, 2010

Certified by .....
Tommi Jaakkola
Professor
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Learning To Reformulate Long Queries

by

Neha Gupta

Submitted to the Department of Electrical Engineering and Computer Science
on May 21, 2010, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science and Engineering

Abstract

Long search queries are useful because they let the users specify their search criteria in more detail. However, the user often receives poor results in response to the long queries from today's Information Retrieval systems. For the document to be returned as a relevant result, the system requires every query term to appear in the document. This makes the search task especially challenging for those users who lack the domain knowledge or have limited search experience. They face the difficulty of selecting the exact keywords to carry out their search. The goal of our research is to help bridge that gap so that the search engine can help novice users formulate queries in a vocabulary that appears in the index of the relevant documents.

We present a machine learning approach to automatically summarize long search queries, using word specific features that capture the discriminative ability of particular words for a search task. Instead of using hand-labeled training data, we automatically evaluate a search query using a query score specific to the task. We evaluate our approach using the task of searching for related academic articles.

Thesis Supervisor: Tommi Jaakkola
Title: Professor

Acknowledgments

I am incredibly grateful to my advisor, Tommi Jaakkola, who gave me the opportunity to work under his guidance. I remember just walking into his office one day and expressing interest in working with him. He immediately followed up and since then I've felt truly blessed to have such a brilliant guide. I was always able to make spontaneous visits to his office and he would instantly make time for our prolonged discussions. Tommi provided valuable insight, helped me make sense through my cluttered reasoning, taught me to look at the big picture and patiently reviewed various drafts of my thesis. His detailed feedback on the write up and posters have made me a better writer and presenter.

I would also like to thank Professor Hari Balakrishnan for first inviting me to work with him at MIT. It was valuable and fun to have stimulating discussions with him on new research topics, and he always encouraged me to chase my interests. I am also grateful to him for letting me use the machine for my research, which made this thesis possible!

I owe a big thanks to some of the members of our group. David Sontag provided frequent and invaluable assistance during the early phase of my research. His guidance and brain storming sessions helped me continue making progress. The visiting post-doc in our group, Patrik Hoyer, was always happy to chat about my research, answering questions and giving me feedback on my posters and write ups.

I am thankful to my family for the sacrifices that they have made to send me to the best schools. They have always stood behind my decisions and encouraged me to pursue what makes me happy and what challenges me. I want to express my gratitude to all my friends at MIT, especially Priya and Sachi, who helped me through tough times, particularly when the flu wiped me out for 2 weeks!

Finally, I am thankful to my fiance, Phil, for his support as we made the difficult decision of spending this time apart. These couple of years would not have been as seamless without his continuous encouragement, prayer and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The explosion of information on the web has made finding relevant information a challenging task. Existing web pages change frequently and new ones constantly appear. The performance of retrieval systems depends heavily on the quality of user interaction. Current systems encourage their users to type in short queries to express their information need. A popular search engine's search help says the following:

> *Describe what you need with as few terms as possible. The goal of each word in a query is to focus it further. Since all words are used, each additional word limits the results. If you limit too much, you will miss a lot of useful information. The main advantage to starting with fewer keywords is that, if you don't get what you need, the results will likely give you a good indication of what additional words are needed to refine your results on the next search.*

Search results retrieved by short (comprising of 2-3 terms) queries are often dominated by those picked up by information cascades, i.e. popular webpages. This poses a serious problem for a domain non-expert [33] trying to find relevant documents specific to their search task. The relevant result could be buried past the first couple of pages of search results. Reading the full text of those results and manually trying various combinations of terms over multiple short search queries becomes an onerous task.

This task of finding the right combination of query terms is easier for some than others. However, search engines provide the same experience to everyone regardless

of their expertise. One has to understand to some extent the mechanics of how a search engine works, in order to formulate effective queries. This deficiency in search engine design has the effect of requiring a longer learning curve for those who are new to the system.

The goal of our research is to help bridge that gap so that the search engine can help novice users formulate queries in a vocabulary that appears in the index of the relevant documents. This is a challenging problem for two key reasons: "vocabulary mismatch" and "incomplete vocabulary".

"Vocabulary mismatch" [32] is a well known problem in Information Retrieval. It arises, for example, when a user lacks the domain knowledge or has limited search experience. As a result, she often has difficulty in selecting exact keywords to carry out a specific search task. Various studies have been done to analyze the importance of search experience and domain knowledge for effective searching on the web [1, 14, 20]. White et. al carried out an extensive study of how web searcher knowledge and strategies differ from each other, and how this can greatly influence their ability to perform successful searches [33]. An example of where domain expertise helps alleviate the problem is illustrated by the fact that it would be easier for a doctor to find the latest research studies related to a particular disease than a computer scientist. Similarly, a user looking for a place to rent in the US will not get the most relevant web pages by searching for "flat rentals". However, an experienced searcher will quickly refine his query to "apartment rentals" finding the term "apartment" to be more discriminative after looking at the first few pages of search results. On the other hand, an inexperienced searcher might give up after the first try and not even consider searching again.

Novice users also face the problem of "incomplete vocabulary" which means that the user is unable to naturally think of additional specific terms to describe the search problem. Attempts to alleviate this problem have relied on automatic term expansion, or the construction of a thesaurus. The drawback of these approaches is that short queries often do not provide enough context and thus term addition in this scenario can lead to changing the original intent of the query. This reduces the number of

relevant documents retrieved by the expanded query.

Query expansion has long been a focus of information retrieval research to help mitigate the above problems. The goal of query expansion is to find and include additional related terms to the original short query to help focus it further. As pointed out earlier, it is mostly useful for addressing the problem of "incomplete vocabulary" for short queries given that the it already includes a few relevant terms. It is not that helpful for long queries since additional terms can drastically hurt recall[1]. Query expansion also does not tackle the problem of improving a poorly formulated query, that is, a query containing mis-specified words. It therefore does not address the "vocabulary mismatch" problem.

Our research focuses on a complementary problem – query *reformulation* which involves query *summarization* and *translation.* We believe that letting the user specify long queries (such as a portion of a text from a webpage) would allow them to specify their search more precisely, with significantly less work on their part. Systems would be able to leverage more contextual information available in the long queries to automatically reformulate it to a more effective and concise version while preserving the original intent.

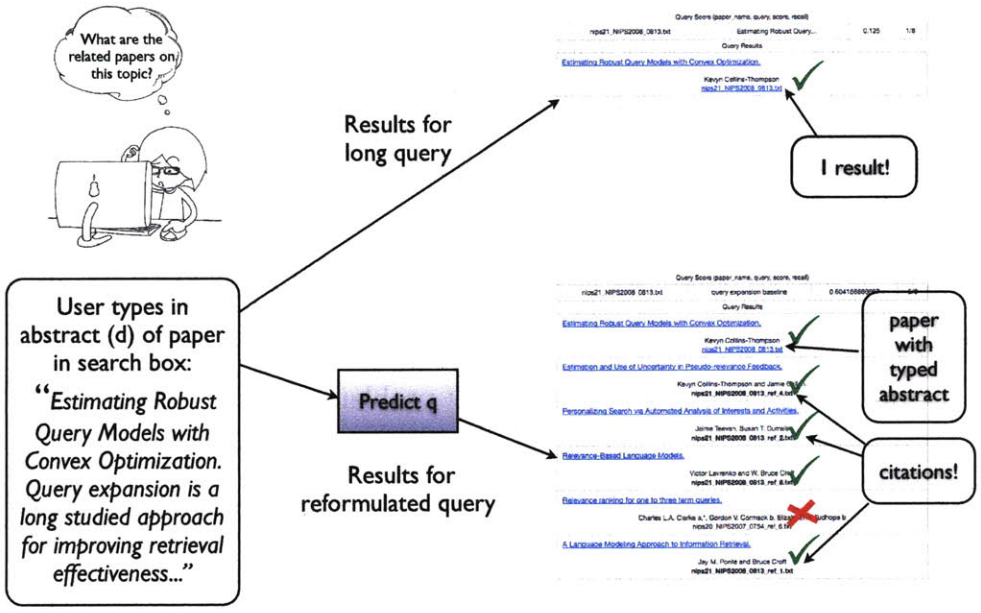
Let us illustrate the utility of query reformulation with an example. Consider the scenario where a graduate student is interested in carrying out an exhaustive literary search for related work for a particular research paper. The student submits the first few lines of the title and abstract

> *Estimating Robust Query Models with Convex Optimization. Query expansion is a long studied approach for improving retrieval effectiveness by enhancing the user's original query with additional related words. Current algorithms for automatic query expansion can often improve retrieval accuracy on average but are not robust that is they are highly unstable and have poor worst case performance for individual queries. To address this problem we introduce a novel formulation of query expansion as a convex optimization problem over a word graph ...*

[1]Defined in Section (4.1)

as a query to our local search engine for a randomly chosen IR paper submitted to a NIPS conference:

Figure 1-1: User Scenario



While the query finds the paper with the above abstract, it contains extraneous terms that resulted in making the query too specific. The query just returns one result which is the paper (nips21_NIPS2008_0183.txt) from which the long query is derived, as shown in figure (Figure 1-2):

Figure 1-2: Original Long Query (d)

Query Score (paper_name, query, score, recall)

nips21_NIPS2008_0813.txt	Estimating Robust Query...	0.125	1/8

Query Results

Estimating Robust Query Models with Convex Optimization.

Kevyn Collins-Thompson
nips21_NIPS2008_0813.txt

Now consider results for a shortened query *query expansion baseline* in figure (Figure 1-3). Compared to the original long query, the set of search results contains a more useful set of related papers. 3 out of top 6 results are the references (nips21_NIPS2008_0813_ref_*.txt) of the paper associated with the long query.

Our research proposes a model to automatically reformulate a seed document (e.g., abstract of a paper) into a more effective search query. For our experiments,

Figure 1-3: Query Reformulation (q)

Query Score (paper_name, query, score, recall)

nips21_NIPS2008_0813.txt	query expansion baseline	0.604166666667	5/8

Query Results

Estimating Robust Query Models with Convex Optimization.

Kevyn Collins-Thompson
nips21_NIPS2008_0813.txt

Estimation and Use of Uncertainty in Pseudo-relevance Feedback.

Kevyn Collins-Thompson and Jamie Callan.
nips21_NIPS2008_0813_ref_4.txt

Personalizing Search via Automated Analysis of Interests and Activities.

Jaime Teevan, Susan T. Dumais
nips21_NIPS2008_0813_ref_2.txt

Relevance-Based Language Models.

Victor Lavrenko and W. Bruce Croft
nips21_NIPS2008_0813_ref_8.txt

Relevance ranking for one to three term queries.

Charles L.A. Clarke a,*, Gordon V. Cormack b, Elizabeth A. Tudhope b
nips20_NIPS2007_0754_ref_6.txt

A Language Modeling Approach to Information Retrieval.

Jay M. Ponte and Bruce Croft
nips21_NIPS2008_0813_ref_1.txt

we work with a corpus of documents consisting of machine learning research papers (from the annual conference "**N**eural **I**nformation **P**rocessing **S**ystems") and their references. A portion of text from each paper (for example, the abstract or the entire document) can be considered a long query for which its references would be the closest relevant documents. We generate the list of reformulated queries by composing terms from the text of the NIPS paper. For example, *query expansion baseline* is a possible reformulation of the long query shown in figure (Figure 1-3). We design a score function to rank the reformulated queries such that queries which return references higher up in the result set receive higher scores. To prevent query drift (this concept is described in [37]), we require that the NIPS paper itself be retrieved by any candidate query. Our approach provides an efficient alternative to manually building a training data set.

Chapter 2

Related Work

Existing work on query reformulation can be roughly divided into three areas.

2.0.1 Query Refinement

A lot of work has been done on query refinement techniques to improve the retrieval performance of queries [10, 8, 9, 16, 12, 34]. This work is focused on improving the quality of a short query and can be broadly classified into two categories. The first category contains Relevance Feedback methods which refine the query based on the initial set of documents retrieved by the user's query. These methods work well if the returned documents are somewhere close to the documents that the user desires. The second category consists of global techniques that carry out query expansion by suggesting additional query terms using a thesaurus such as WordNet, or by building a thesaurus automatically by mining search query logs. Such approaches suffer from query drift because the queries are too short and do not provide enough context.

The work by Wang and Zhai [32] presents a probabilistic approach to analyzing the relations between terms in a query. Their work mines the co-occurrences of terms within multi-word queries in query logs and uses those statistics for proposing a more discriminative term in place of an original term or suggests a term for expanding the original query. Their strategy is to replace a term at a time and get a possible list of candidate reformulations which differ from the original query by only one term.

Then they recommend the most probable candidate as a substitution. The translation model collects the candidate terms for substituting an original term based on whether the two appear in similar contexts. The authors use global information present in search query logs to mine term association patterns rather than restricting themselves to a single user session.

The focus of the above work has been to refine short queries by adding or substituting query terms. In contrast, our work aims to summarize very long queries.

2.0.2 Query by Document

Yang et. al [35] present the idea of querying by document (QBD) to find content related to the document. They propose a heuristic using part-of-speech tagging to extract a set of *noun phrases* from the document which are then used to construct queries to a search engine. The authors evaluate the quality of their phrase extraction technique and document retrieval using human judges, limiting the number of query documents to 34. Obtaining large amounts of manually labeled training data is expensive and often infeasible. Their QBD-WI approach maintains a graph of 7 million nodes which takes roughly 10 seconds to list the queries for a document. This is an expensive procedure to run for a user query in real time. However, using Wikipedia to generate additional query terms is an interesting idea.

The authors of [4] present a class of non-linear models based entirely on words. The proposed low rank approximation method is not only computationally efficient but also captures higher order relationships between words. Their evaluation uses whole Wikipedia articles as queries and thus can be applied to query by document scenario. They consider the task of ranking highly the documents that the input Wikipedia article links to. One problem with this approach is that given an input document as a query, the relevance scores for all documents in the corpus need to be computed.

The work by Guo et. al [18] falls into the same category applied to patent prior art search. Given a new patent as a long query, their work looks into identifying a list of previously granted patents to be useful to include as patent citations. They

derive patent structure specific features for the task of learning patent ranking.

Yih et al. [36] present a paper on automatically extracting keywords from the text of webpages for displaying advertisements. They use a variety of linguistic features such as whether the word appears capitalized, as a noun phrase, in the title etc. coupled with IR features such as word frequency. More interestingly, they use a query log feature based on the frequency of the occurrence of the word/phrase in query logs, which they find to be quite informative. This paper might not seem to be directly related to our work but the feature set can certainly be useful for extracting discriminative terms from the document which can then be combined for query reformulation task.

In our work, we automatically generate a large training and test set by using any corpus of text documents. This enables us to consider machine learning approaches which can *learn* how to best query by document. The approach is based on query reformulation. Once the parameters are learned by the model, query reformulation for a new document is extremely efficient. We used a motivation similar to [18] to construct our data set. The title and abstract of a paper can be seen as a sample description from the complete discourse available in the related papers (for example, the references) that the user is interested in finding. Our goal is to find a way to extract the most discriminative terms describing the discourse on the basis of the sample provided by the user.

2.0.3 Query Reduction

Many approaches have already been suggested for formulating shorter queries from long ones. Work by [24] set the stage by looking at rewriting the query to a version that is comprised of a small subset of appropriate terms in order to improve query effectiveness. They select a set of top ranked sub-queries using a single feature of mutual information (MI) and presented them to the user.

Bendersky and Croft in [5] approach the problem of handling more verbose natural language queries by presenting a way of identifying the key concepts from them. They extract noun phrases from the query as concepts and present a way of weighing them

using a supervised machine learning approach. They use corpus-dependent features such as *tf*, *idf* of the phrase, query-dependent feature which measures the frequency of a concept in the query log and corpus-independent feature such as Google *n-grams* to estimate the frequency of terms appearing in a concept in a large web collection. Next they use the highest-ranked concepts for each query to improve the retrieval effectiveness of the original query.

Similarly, [25] presents a machine learning technique to automatically carry out query reduction by picking the top-ranked sub-query in order to improve relevance. Their approach uses a limited set of hand-constructed features during learning. Nevertheless, these features can be applied to our problem, and we will use the approach of [25] as the baseline to compare our method's performance.

Chapter 3

Experimental Setup

3.0.4 Corpus

We use a collection of Nips papers [1] and the papers mentioned in their citations as our corpus of documents. Standard data sets like TREC provide a very small collection of queries and often do not provide the actual text of all the documents. So we constructed our corpus of 1156 NIPS papers which we used to generate long queries. In order to download the references for each NIPS paper, we parsed the set of citations that appeared in each NIPS text file. The first few words from each citation were used as a query to a popular search engine. We downloaded the file if the first search result contained a link to a pdf file. All of the pdf files were then converted to text using the pdftotext utility. We manually evaluated the quality of our corpus to find that the number of correct references for a few randomly chosen NIPS papers was nearly or more than 50%. We removed the NIPS documents with less than four total references. After the cleanup, we were left with 1156 papers and the total size of the corpus was 8927 documents. Note that the document collection contained some duplicates for papers with common references.

[1] http://books.nips.cc/

3.0.5 Search Engine

We use the indexing and retrieval capabilities of Apache Lucene [2] which is a high-performance, text search engine library written in Java. It uses *tf-idf* based scoring to compute the relevance scores for each result. The exact function used by Lucene to compute the relevance score of result r for query q is given by:

$$score(q, r) = \sum_{t \in q} tf(t \in r).idf(t) \tag{3.1}$$

where $tf(t \in r)$ is the term frequency for the query term t in result r and $idf(t)$ is just the inverse document frequency of the term. We changed the default operator for parsing queries from OR to AND so that a match is found only if all the terms exist anywhere in the text of the document.

3.0.6 Web Interface

We built a web interface on top of Lucene using web.py [3]. Querying the NIPS corpus by typing in a query into the search box (Figure 3-1) makes it an extremely useful debugging tool.

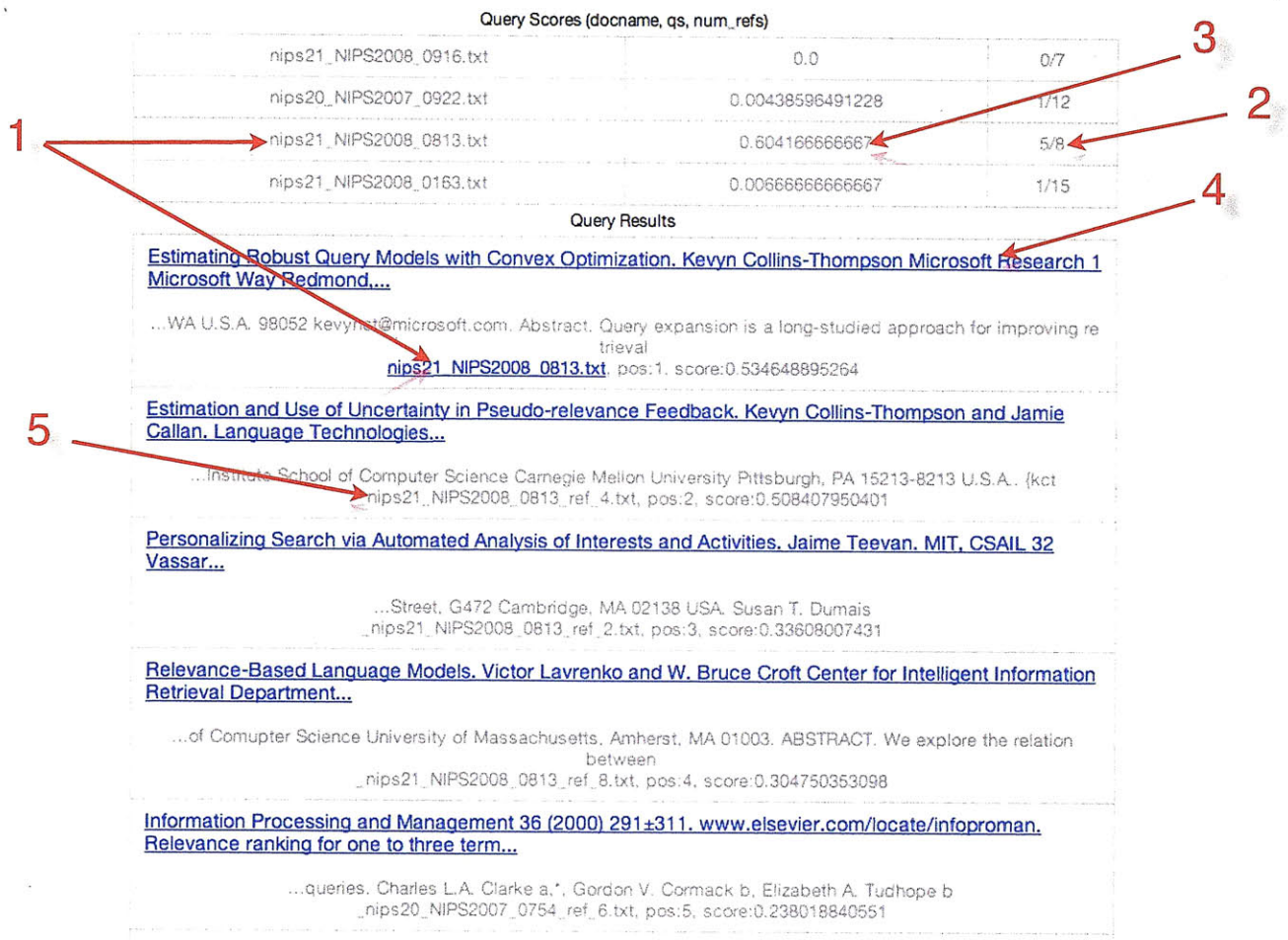
Figure 3-1: Web Interface

Figure (Figure 3-2) below shows the results for the query once the user hits the search button. The diagram below points out five main features useful for understanding query quality:

1. NIPS paper: One of the NIPS papers fetched by the search query. This is one of the 1157 papers and cannot be a reference, i.e. it does not contain the string

[2] http://lucene.apache.org/java/docs/index.html
[3] http://webpy.org/

Figure 3-2: Search Result Page



"*_ref_*" in its name. When present as a search result, its name links to the full text of the paper.

2. Related papers fetched: Shows the number of related papers fetched by the query for the NIPS paper specified in the first column.

3. Query score: Shows the query score for the NIPS paper specified in the first column.

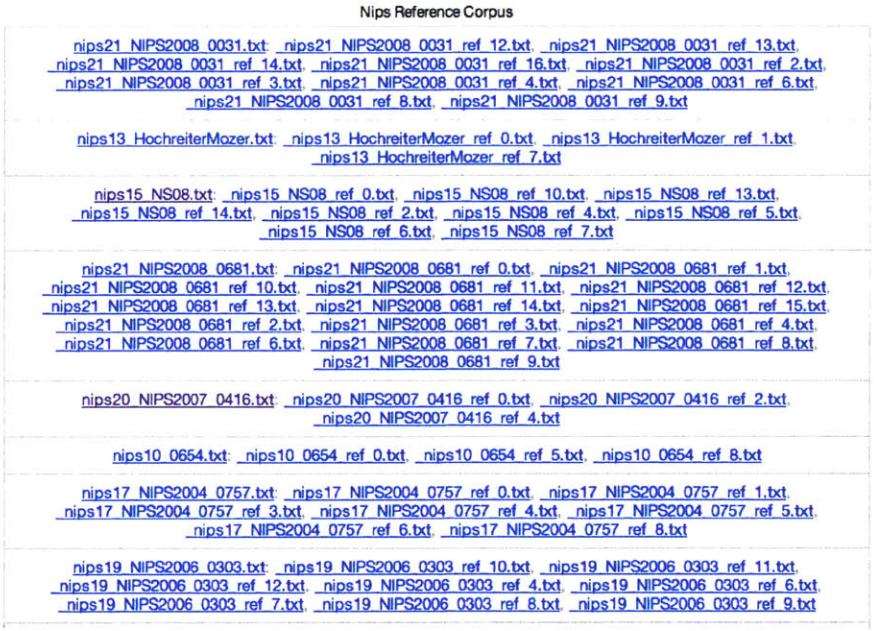
4. Title: The first line of the search result is the title of the paper.

5. Reference papers: The last line contains the name of the paper returned. If it

is not a link, then it is a reference of a particular NIPS paper. The naming convention allows us to quickly tell which NIPS paper cites it as a reference.

In addition to the main search interface, various url handlers were added as needed, to quickly visualize results of a particular model or to analyze new features. Figure (Figure 3-3) shows a portion of one such page which lets the user browse the whole NIPS corpus. Each row contains a link to the full text of a NIPS paper followed by links to its references.

Figure 3-3: Nips Corpus



Chapter 4

Discriminative Approach

We are interested in the general problem of learning a mapping from a long search query to a short reformulated query. The task is challenging since a good query reformulation involves both query *reduction* and *translation*. The challenge is to pick the "right" combination of terms that are neither too specific nor too general while tailoring the query to the user's domain of interest.

The mapping that we learn is represented by a ranking function that evaluates each reformulated query in relation to the long query. The function is learned discriminatively from a training set so as to reproduce a query quality score, evaluated with full knowledge of the desired results. This query score is used as a measure of the quality of a reformulated query in order to prefer certain query reformulations over the others.

4.1 Query Score From Search Engine

This section describes a method for computing the query score for evaluating the quality of a query. Assuming we know the desired results for the long query, we run a candidate query through the search engine. A numerical score is assigned to the query based on the results retrieved from the index. The score for a query is given by its **A**verage **P**recision value.

In Information Retrieval, precision and recall are two widely used statistical clas-

Figure 4-1: Query Score

sifications. **Precision** is defined as the number of relevant documents retrieved by a search divided by the total number of documents retrieved by that search. **Recall** is equal to the number of relevant documents retrieved by a search divided by the total number of relevant documents in the index for that search. AP is a function of both precision and recall and is obtained by averaging the precision values at each relevant result observed in the returned list as follows:

$$
\begin{aligned}
q &= \text{Query} \\
N &= \text{Total number of retrieved results considered} \\
R &= \text{Set of relevant documents in the index for } q \\
rel[i] &= \begin{cases} 1 & \text{i'th result is in set } R \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

$$
\begin{aligned}
P@j &= P[j] = \sum_{k=1}^{j} rel[k]/j \\
S(q,R) &= \sum_{j=1}^{N} (P[j].rel[j])/|R|
\end{aligned}
$$

The query score $S(q, R)$ prefers queries that return the relevant results earlier in the list of retrieved results. As an illustrating example, figure (Figure 4-2) shows the query score computation for the long query in (Figure 1-2) and one possible query reformulation in (Figure 1-3). It is evident that the query score for the reformulated query q is 0.57 which is higher than that of the long query d. The score is computed

carefully to not count duplicates more than once.

AP gives us query score for a single query. For our experiments, we will be using **M**ean **A**verage **P**recision as the evaluation metric. MAP is just the mean of individual query scores.

Figure 4-2: **AP** Computation

Docs	D1	D2	D3	D4	D5
j	1	2	3	4	5
rel[j]	1	1	1	1	0
P@j	1	2/2	3/3	4/4	4/5

|R| = 7
S(q, R) = **A**verage **P**recision
= (1 + 2/2 + 3/3 + 4/4) / 7
=0.57

Similarly, S(d, R) = 0.14

4.2 Learning Goal

Suppose we are given an input long query d, a universe of all possible reformulated queries U and a set R of relevant results. The goal during learning is to learn a function that would rank the reformulated queries as the query score would rank them. There are two problems with this: the query scores are available only during training and the universe U is very large. So a parametric mapping is learned from a smaller candidate set of queries Q associated with each d. We describe how Q is constructed in Section 4.3. Once we have the parameters, we can rank the candidate queries for any new long query without access to the query scores.

4.2.1 Learning Criteria

Given a training set $T_n = \{(d_i, R_i, Q_i)\}_{i=1,\ldots,n}$, we learn the parameters of θ by training on all pairwise preferences between queries of each set Q_i. Q_i represents the query space that we are interested in evaluating for the long query d_i.

The objective function that we need to minimize is:

$$\frac{1}{2}\|\theta\|^2 + \frac{C}{n}\sum_{i=1}^{n} \sum_{q_j,q_k \in Q_i \text{ s.t. } S(q_j,R_i)>S(q_k,R_i)} \xi_{ijk} \tag{4.1}$$

subject to the following constraints, $\forall i \in \{1, \ldots, n\}$:

$$\forall q_j, q_k \in Q_i \text{ s.t. } S(q_j, R_i) > S(q_k, R_i),\ \xi_{ijk} \geq 0$$

$$f(d_i, q_j; \theta) - f(d_i, q_k; \theta) \geq \triangle S(q_j, q_k, R_i) - \xi_{ijk} \tag{4.2}$$

where

$$\triangle S(q_j, q_k, R_i) = S(q_j, R_i) - S(q_k, R_i)$$

Given the relevant result set R for query q, $S(q, R)$ represents the query score for q as evaluated by the search engine. The above constraints state that the score for q_j must be greater than the score of q_k by a required margin, for each d_i. This margin is equal to the loss $\triangle S(q_j, q_k, R_i)$ which means that we require larger margin for pairs of queries with large differences in their query scores as evaluated by the search engine.

4.2.2 Predicted Query

This section lists a couple of different ways by which one can derive a predicted reformulated query during test time.

Top Ranked Query

Given a new long query d and a candidate set of reformulated queries Q, one can rank the queries in the candidate set using the learned ranking function and output the topmost one.

$$\hat{q} = \arg\max_{q \in Q} f(d, q, \theta)$$

We will refer to the predicted query output using this operation as *top_q*.

Query Composed of Top Ranked Words

The above method assumes the availability of the candidate set of reformulated queries for each new long query. It is very likely to not be the case in practice. For instance, it might be computationally too expensive to derive all candidates from a user's input query. However, it is often possible to propose a method for deriving the predicted query just from the learned parameters and a new long query. As an example, we know that $f(d, q, \theta)$ assigns a ranking score to query q. One can compute a similar score for each candidate word in the vocabulary and order them by their scores. A simple heuristic for outputting the query can be to just concatenate top ranked candidate words together. A query output using such methods will be referred to as *argmax_q*.

4.2.3 Evaluation Metric

In order to evaluate the performance of the trained models, we compute the average of the search engine scores of *top_q* and *argmax_q* over the long queries in training/test/validation data. Mean Average Precision serves as the primary metric with scores measured in terms of P@20.

4.3 Query Generation

For each NIPS paper, the abstract and title give us the long query. Next, we generate a candidate set of reformulated queries for each long query. We use the following intuitions as guidelines for selecting terms for candidate reformulated query set:

1. Stop words are too general to be useful to be a part of good query.

2. The abstract of the paper is a self-contained and short statement that describes the rest of the paper. The abstract incorporates key words found in the text and often includes purpose, methods and the scope of research. As such, it

also contains terms that refer to the approach used in prior related work. Thus the terms in the abstract are potentially useful for being part of a reformulated query.

3. Carrying out search using the long query only returns the paper with the exact abstract and title from which the long query is derived. No other document in the index contains all the terms present in the long query. Thus, the query score for a long query is always $1/R$ where R is the number of relevant documents for the long query in the index. The score of reformulated query should be at least $1/R$ in order to not hurt performance. A query composed of high *idf* terms results in low recall since by definition those terms are present in very few documents. Usage of such terms, however, helps in maintaining high precision. At worst, only one document will be retrieved giving us the score of $1/R$. The challenge lies in picking the combination of terms that are not specific to just a few documents in order to improve recall while keeping precision high. So we seek reformulations that involve high *idf* terms to maintain precision while also searching for combinations that increase recall.

Given the above insights, we consider the set of high *idf* terms that appear in the long query as the candidate set of query terms for the reformulated query. In addition, we remove terms from the vocabulary that only appear once in a particular paper to get rid of any misspellings. Previous research on query logs [29] has shown that the average length of the query is about 2.35 terms. Azzopardi's analysis [3] also concludes that the most effective queries contain between two to five terms. So we generate reformulated queries of lengths one, two and three for a given document. The above procedure is used to generate two different data sets. The statistics of the data sets are illustrated in Table (4.1). The second column displays the total number of queries across all documents with score greater than 0.

The following sections contain details specific to each data set.

Table 4.1: Dat Set Statistics

Data Set	Num Docs	Num Queries
Data Set I	1156	1,559,590
Data Set II	1156	1,568,778

4.3.1 Data Set I

The relevant set of documents for each query consists of the references of the paper from which the corresponding long query is extracted and the paper itself. Sample queries for the long query in figure (Figure 1-2) are provided in table (4.2).

Table 4.2: Sample query reformulations for long queries in figure (Figure 1-2).

Original query: *Estimating Robust Query Models with Convex Optimization. Query expansion is a long studied approach for...*		
Data Set I	**AP**	**Recall**
query baseline expansion	0.604	5/8
query improving word	0.589	5/8
query retrieval expansion	0.589	5/8
query expansion word	0.578	5/8
query baseline improving	0.578	5/8
collections baseline feedback	0.284	3/8
...	...	...
retaining query aspect	0.125	1/8
gains effectiveness word	0.025	1/8

4.3.2 Data Set II

For some papers, the set of correct references that we could download or the number of references cited in the paper was too small. As a result, the top queries for such papers are composed of terms too specific to the vocabulary of one or two references that can be retrieved. Table 4.3 provides an example. To make the link structure within the corpus more interesting, we computed a tf-idf based cosine similarity score for a paper with every other document in the corpus. We augmented the set of relevant documents for the paper with documents having a similarity score above a fixed threshold. This change did not have much effect on the top queries for papers with decent number of

references. For instance, the queries for long query in figure (Figure 1-2) stayed about the same. However, the quality of top query reformulations improved for other papers. Table 4.4 seems to have better set of top query reformulations for the same long query presented in table 4.3. The top queries in table 4.3 consist of terms specific to couple of references containing details about handwritten digit recognition and DNA splice site detection.

Table 4.3: Data Set I Example Query Reformulations

Original query: *Thin Junction Trees. We present an algorithm that induces a class of models with thin junction trees— models that are characterized by an upper bound on the size of maximal cliques... We illustrate this approach with applications in handwritten digit recognition and DNA splice site detection.*		
Data Set I	**AP**	**Recall**
splice handwritten	0.333	2/6
junction handwritten	0.333	2/6
ensuring digit	0.333	2/6
splice tractable	0.333	2/6
dna handwritten trees	0.333	2/6
...	...	...
junction trees	0.075	2/6
cliques junction trees	0.055	1/6

But the focus of the paper is surrounding thin junction trees rather than its applications towards handwritten digit recognition or DNA splice site detection.

4.4 Baseline Model: QP

QP refers to the model that we use for baseline comparison. It is trained using SVM^{rank} with the Query Predictors proposed in [25]. The set of query quality predictors from the paper that we use as features are described below.

SQLen

Number of terms in the query.

Table 4.4: Data Set II Example Query Reforumlations

Original query: *Thin Junction Trees...*		
Data Set II	**AP**	**Recall**
junction trees	0.4567	8/17
junction maximal trees	0.3445	6/17
cliques junction	0.29411	5/17
junction tractable trees	0.2352	4/17
cliques thin trees	0.1765	3/17
...	...	...
splice handwritten	0.1176	2/17
ensuring digit	0.1176	2/17

MI

Explained in Section 4.6.1.

IDF-based features

IDF of each query term is calculated as:

$$IDF_w = \frac{\log_2 \frac{N+0.5}{N_w}}{\log_2(N+1)}$$

where N_w is the document frequency of term w and N is the number of documents in the corpus. We calculate the (a) sum (b) standard deviation (c) maximum/minimum (d) maximum (e) arithmetic mean (f) geometric mean (g) harmonic mean and (h) coefficient of variation of the IDFs of constituent words and include them as features as suggested by the paper.

SCQ

We avoid the expensive computation of query clarity (QC) and instead used the simplified clarity score (SCQ) as a comparable pre-retrieval performance predictor. SCQ for each query term is given by:

$$SCQ_w = (1 + \ln \frac{T_w}{N}) \ln(1 + \frac{N}{N_w})$$

T_w is the number of times w appears in the entire corpus. Aggregate values similar to the ones presented in Section 4.4 are included as query quality predictors.

ICTF

Inverse **C**ollection **T**erm **F**requency of term w is defined as:

$$ICTF_w = \log_2 \frac{T_w}{T} \tag{4.3}$$

where T is the number of term occurrences in the collection. Using the ICTF values of each query term, aggregate statistics listed in Section 4.4 are incorporated as features.

Given the above set of query quality predictors as features, we train a linear classifier and pick the best value for regularization parameter C by its performance on the validation set. Implementation details can be found in Section 4.5.3.

4.5 Feature Set I: TM

We begin by presenting a very basic Translation Model based on word-to-word mappings. This learning scenario is based on the intuitive notion that the model can learn to substitute a more discriminative word in place of the one specified by the user in the search query. A feature z: $\mathcal{U}X\mathcal{V} \rightarrow \{0,1\}$ is defined for every pair of words in the long query d and reformulated query q as follows:

$$z_{uv}(d,q) = \begin{cases} 1 & \text{if } u \in d,\ v \in q \\ 0 & \text{otherwise.} \end{cases}$$

where $\mathcal{U}$ and $\mathcal{V}$ is the vocabulary of long queries and reformulated queries respectively. Word pairs that do not appear sufficiently often in the training data are removed from the vocabulary. Filtering of the reformulated query set based on this heuristic highlights a limitation of this model. The learning paradigm is never able to predict certain set of top queries containing rare but potentially useful words from a held out long query.

The reformulation from d to q is governed by parameters θ_{uv} where $u \in \mathcal{U}$ and $v \in \mathcal{V}$. The reformulation score from d to q is then given by:

$$f(d,q;\theta) = \vec{z}(d,q).\vec{\theta} = \sum_{u \in d} \sum_{v \in q} \theta_{uv}$$

Now that the TM model is fully specified, the model parameters are estimated using the learning criteria specified in Section (4.2.1).

4.5.1 Query Composed of Top Ranked Words

During test time, we carry out query reformulation using our model by calculating the $argmax\_q$. Given the learned parameters θ_{uv} and a new long query d, we find the optimum q by computing a score for each candidate query term v:

$$score(v) = \sum_{u \in d} \theta_{uv},\ \forall v \in \mathcal{V} \tag{4.4}$$

We concatenate the top 3 scoring v's to get a query of length 3.

There are several advantages of using this approach. It doesn't require a set of possible reformulations available for a long query during test time, making the prediction task quite fast. We can also incorporate this approach during the training phase to expand the set of reformulated queries for a particular long query. Having a well-defined query score and a local search engine lets us evaluate the quality of the newly generated query quickly. The next section describes how this test time procedure can be integrated in the training phase to improve the quality of $argmax\_q$ iteratively.

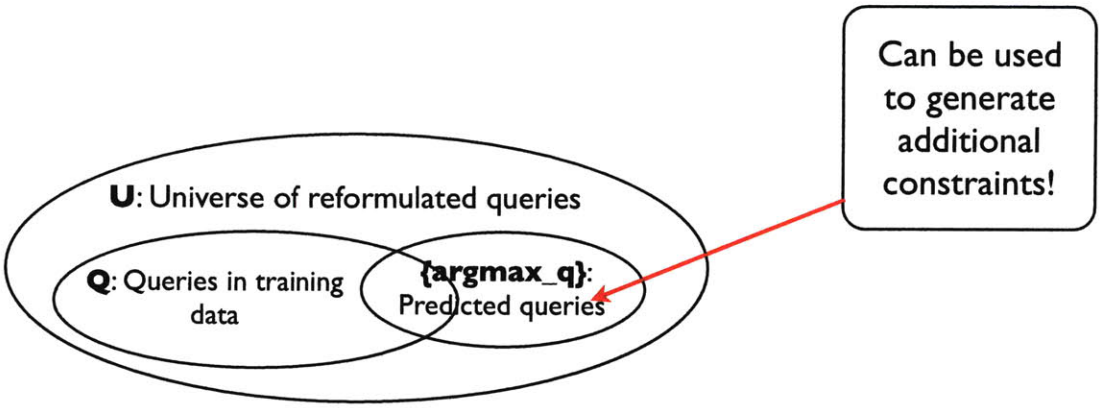
There are limitations of using such a simple heuristic. One such limitation is that the derived queries do not consider interactions between the chosen query words. This could lead to picking a query word which does not improve query quality and is in fact redundant to the word already chosen. "beliefs mixture mixtures" is a simple example that shows such redundancy in the $argmax\_q$ that the model output for one of the test queries. We present a model extension in section 4.5.4 as one way of capturing interactions between query terms.

4.5.2 Cutting Plane Algorithm

We propose a novel algorithm that enables us to learn the parameters efficiently despite a large number of possible constraints. Our training algorithm picks a restricted set of reformulated queries for each long query as a hypothesis space. Once the parameters are learned, the training performance is assessed by computing the $argmax\_q$ for each long query in the training data. The computed $argmax\_q$ can lie outside the training set as shown in figure (4-3) which can be used to add additional ranking constraints. Figure 4-4 presents the sketch of the algorithm.

The algorithm starts with an initial set of constraints for each d given by queries in Q. It iteratively expands this set by computing the query score for $argmax\_q$ as returned by the search engine and adding it to the set of constraints if $argmax\_q$ is a new reformulated query. Incorporating additional constraints corresponding to new queries is indeed beneficial. Adding pairwise ranking constraints for the new

Figure 4-3: Ranking Constraint Set Expansion for Training Pair (Q, d)



predicted query ensures that the queries predicted by the model with high query score are indeed ranked highly by the model in the training set as well. Similarly, the model can learn from poorly performing predicted queries by forcing it to rank them lower than the others in the training set. Thus this mechanism lets us start with a small set of candidate queries for each long query and generates queries on the fly while also improving the quality of $argmax\_q$.

Inputs:
Training data $T_n = \{(Q_i, R_i, d_i)\}_{i=1,\ldots,n}$

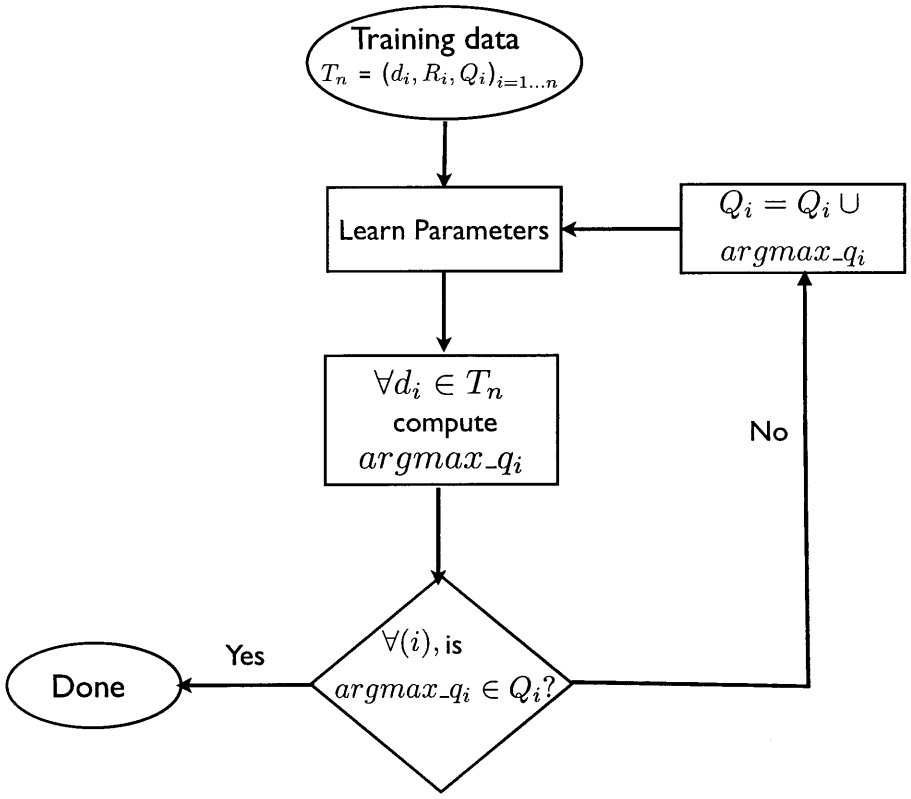
Algorithm:

1. Let θ^* be the solution of equation (4.1) on T_n.
2. For each d_i:
   (a) Compute $argmax\_q$ using (4.4).
   (b) Search using $argmax\_q$ and compute its query score.
   Is $argmax\_q \in Q_i$? If not, $Q_i = Q_i \cup argmax\_q$.
3. Go to Step 1

Figure 4-4: Cutting Plane Algorithm

Pictorially, the algorithm proceeds as shown in figure (4-5).

Figure 4-5: Cutting Plane Flowchart



4.5.3 Evaluation

We use a learning-to-rank algorithm as explained in [22] to rank top 1000 queries for each long query consistent with their ordering as provided by the search engine. All the evaluations for this feature set are done using Data Set I. The model only considers a restricted set of possible query words so as to keep the number of parameters small. The candidate queries extracted from the long query were filtered accordingly. We downloaded the SVM^{rank} implementation as defined in [21] and used Linear kernel to train our model. The loss function was set to the total number of swapped pairs summed over all queries. The optimal C value was chosen using the validation set. The size of the feature set was approximately 45000.

Performance Metric

Average *top_q* and *argmax_q* predicted scores values are computed to evaluate our experiment results. Mean Average Precision serves as the primary metric with scores measured in terms of P@20.

Baseline Comparison

We compare the performance of TM model against the baseline model QP presented in section (4.4) and a *High IDF* query generated by concatenating 3 randomly chosen high *idf* terms from the long query. Tables (4.5) and (4.6) present the results on test data.

Table 4.5: **Train**: TM Baseline Comparison Data Set I

Train		
Model	oracle_qs	top_qs
High IDF	0.54	0.15
QP	0.54	0.34
TM	0.54	0.30

Table 4.6: **Test**: TM Baseline Comparison Data Set I

Test		
Model	oracle_qs	top_qs
High IDF	0.55	0.15
QP	0.55	0.23
TM	0.55	0.22

Notice that the training performance of QP is better than TM since it ranks the unfiltered set of top 1000 training queries but the test performance on the entire test set is about the same for both models.

Argmax Query Performance

Next we look into the quality of *argmax_q* for the training and test data sets after the first iteration of the Cutting Plane Algorithm. It is evident from the experiment

Table 4.7: **Train**: TM *argmax_q* Quality

Train		
C Value	oracle_qs	top_qs
0.001	0.54	**0.05**
0.01	0.54	0.03

Table 4.8: **Test**: TM *argmax_q* Quality

Test		
C Value	oracle_qs	top_qs
0.001	0.55	**0.023**
0.01	0.55	0.02

results in Tables (4.7) and (4.8) that the quality of the queries generated using (4.4) is quite poor.

The translation model often outputs reformulated queries composed of words not present in the long query. This leads to poor performance because of several possible reasons. First, the set of relevant documents for some NIPS papers in Data Set I is small (~2-3). Second, the search engine is extremely unforgiving with the requirement that all the query terms need to be present in the text of the retrieved results. The combined effect of the above often results in queries with 0 scores. Third, the training data is probably too small to confidently learn parameter values for word-to-word mappings leading to overall poor quality of *argmax_q*. Nonetheless, the quality can be improved a bit using the Cutting Plane Algorithm. The performance results are presented in the next section.

Cutting Plane Algorithm Evaluation

This section presents the results of the iterative training algorithm. The model ranks top 1000 queries in the test data for outputting *top_q* which explains the higher value of *top_qs* as compared to Table 4.6. The quality of *argmax_q* improves as the algorithm progresses, as shown in tables Tables (4.9) and (4.10).

In order to help with analysis, we divided the type of *argmax_q* into 3 categories as listed in Table (4.11).

Table (4.12) contains the sample output of the collected statistics for a few long

Table 4.9: **Train**: TM Cutting plane results

Train			
Iteration number	oracle_qs	argmax_qs	top_qs
0	0.54	0.021	0.30
2	0.54	0.076	0.30
16	0.54	0.13	0.30
36	0.54	**0.14**	0.30

Table 4.10: **Test**: TM Cutting plane results

Test			
Iteration number	oracle_qs	argmax_qs	top_qs
0	0.55	0.01	0.25
2	0.55	0.033	0.25
16	0.55	0.05	0.25
36	0.55	**0.05**	0.25

queries/documents. *argmax_qs* refers to the search engine score of *argmax_q*.

The total number of *argmax_q* and **argmax_q* across all the documents give us the number of new examples that are added per iteration. Fig (4-6) plots the count of each type of predicted query and the number of examples that are added per iteration. It is evident from the figure that the number of *argmax_q*'s with a non-zero score increases as the cutting plane algorithm progresses but the examples become more and more redundant. Such analysis can help us determine an optimum stopping criterion for the algorithm.

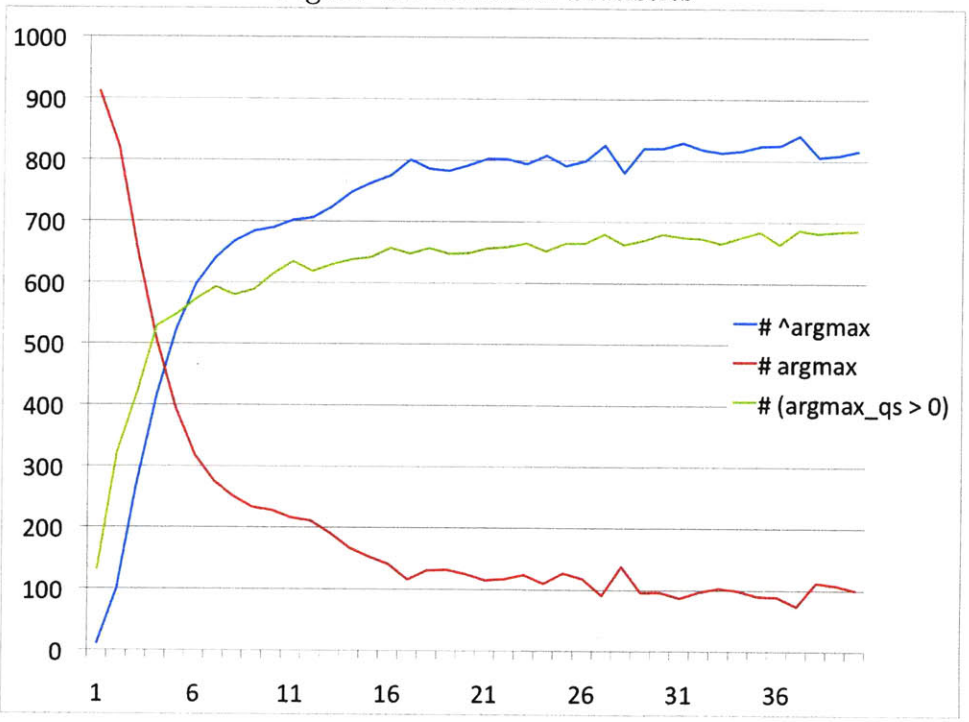
241 new top scoring *argmax_q* were discovered and a total of 8502 new unseen *argmax_q* were added to the training set by the end of iteration 36. On average, the reformulated query set for each long query contained around 10 new queries. The improvement in the quality of *argmax_q* is a promising result of using the cutting plane algorithm to train our model. Table (4.13) shows an example where the algorithm generated a top scoring *argmax_q* which was not available in the original training set for the document.

There is certainly room for improvement in the quality of *argmax_qs* but the results show that the cutting-plane algorithm helps in closing this gap.

Table 4.11: Argmax Query Types

Type	Description
$argmax\_q$	New unseen query added to the training set for the next iteration.
$^{\wedge}argmax\_q$	Query already present in the training set for the document. Ignored.
$^{*}argmax\_q$	New unseen highest scoring query for the document.

Figure 4-6: Iteration Statistics



Modified Objective

The motivation behind modifying objective (4.1) is that we use ranking loss during training to rank all the queries but are only interested in the highest ranked query. By forcing the model to capture the relative ordering of mid-range queries, we may actually be steering it away from ranking the top few correctly. So rather than requiring every pair of queries to be ordered correctly, we modify our objective to require that each individual query is ordered correctly with respect to the top query. The modified objective (slack-rescaling) can be written as:

$$\frac{1}{2}\|\theta\|^2 + \frac{C}{n}\sum_{i=1}^{n}\xi_i$$

Table 4.12: Cutting Plane Iteration Statistics

iteration num	*doc name*	*argmax_q*	*argmax_qs*
0	nips10_0329.txt	stability excitatory synapse	0.0
	nips10_0350.txt	agent movements iteration	0.0
	nips10_0378.txt	energy mistake fully	0.0
	nips10_0486.txt	bottom predictive hebbian	0.0
	nips10_0549.txt	hebbian agent connectivity	0.0
	nips10_0626.txt	em principal nonparametric	0.0
	nips10_0654.txt	predictive perceptron cortical	0.0
	nips10_0787.txt	background target face	0.0
	nips11_0038.txt	iteration synapse curves	0.0
	nips11_0111.txt	timing recordings redundancy	0.0
	...	...	...
1	nips10_0329.txt	stability excitatory upper	0.2
	nips10_0350.txt	iteration latent hinton	0.0
	nips10_0378.txt	synapse energy fully	0.167
	nips10_0486.txt	eye bottom hebbian	0.086
	nips10_0549.txt	hebbian connectivity batch	0.0
	nips10_0626.txt	^em principal covariance	0.5
	nips10_0654.txt	cortical trained robust	0.0
	nips10_0787.txt	surface background target	0.0
	nips11_0038.txt	iteration curves word	0.0
	nips11_0111.txt	^recordings timing redundancy	0.0
	...	...	...
12	nips10_0329.txt	*stability excitatory feedback	0.367
	nips10_0350.txt	^hinton minima boltzmann	0.33
	nips10_0378.txt	^synapse energy fully	0.167
	nips10_0486.txt	^bottom hebbian factor	0.014
	nips10_0549.txt	^hebbian rl batch	0.083
	nips10_0626.txt	^em principal covariance	0.5
	nips10_0654.txt	^objects orientation trained	0.0
	nips10_0787.txt	^surface shapes people	0.1
	nips11_0038.txt	curves iteration spike	0.0
	nips11_0111.txt	^recordings redundancy timing	0.0

Table 4.13: Example of *argmax_q* quality improvement for a single document

iteration num	argmax_q	argmax_qs
0	principal rank analyzers	0.2
1	principal rank latent	0.0
2	principal rank ica	0.0
3	*principal rank mixture	0.6
4	*principal rank mixture	0.6
5	*principal rank mixture	0.6

s.t.

$$\forall i,\ \forall q \in Q \backslash q_i^* : f(d_i, q_i^*; \theta) - f(d_i, q; \theta)) \geq 1 - \frac{\xi_i}{\triangle S(q_i^*, q, R_i)}$$

where

$$\xi_i = \max \left\{ 0,\ \max_{q \neq q_i^*} \{ \triangle S(q_i^*, q, R_i)(1 - f(d_i, q_i^*; \theta) + f(d_i, q; \theta)) \} \right\},\ q \in Q_i \qquad (4.5)$$

The above formulation can be seen as a structured prediction task [31] where the problem is to learn a mapping from input queries d to a reformulated query $q^* \in Q$ based on a training sample of input-output pairs $T_n = \{(d_i, R_i, Q_i)\}_{i=1,\ldots,n}$. The elements of Q are structured objects instead of a scalar response variable. We implemented cutting plane method for training structural Support Vector Machine as described in [23] using SVM^{python}, which exposes a Python interface to SVM^{struct} [1]. The most violated constraint [23] is computed as shown in 4.5.

This method gives us a couple of advantages. It reduces the number of constraints from quadratic to linear making it feasible to train on larger data sets. We also noticed that the training time for RankSVM becomes prohibitive as the value of C is pushed higher. This algorithm let us run experiments with high C values.

It is evident from the results in tables 4.14 and 4.15 that our model performs very well on training data. However, the current feature set clearly leads to over fitting and generalizes poorly given the small amount of training data.

[1] http://svmlight.joachims.org/

Table 4.14: **Train**: TM Modified Objective

Train			
C Value	oracle_qs	top_qs	argmax_qs
0.0001	0.54	0.178	0.001
0.01	0.54	0.22	0.033
1	0.54	0.23	0.024
100	0.54	0.28	0.04
1000	0.54	0.35	**0.06**
1000000	0.54	**0.43**	0.013

Table 4.15: **Test**: TM Modified Objective

Test			
C Value	oracle_qs	top_qs	argmax_qs
0.0001	0.55	0.183	0.0007
0.01	0.55	0.17	0.01
1	0.55	0.18	0.01
100	0.55	**0.21**	**0.01**
1000	0.55	0.18	0.006

4.5.4 Extension: word assignment model

This section explores a model enhancement based on the observation pointed out in section 4.5.1. The model fails to capture any interactions between the chosen reformulated query words. To address this, we suggest the following approach.

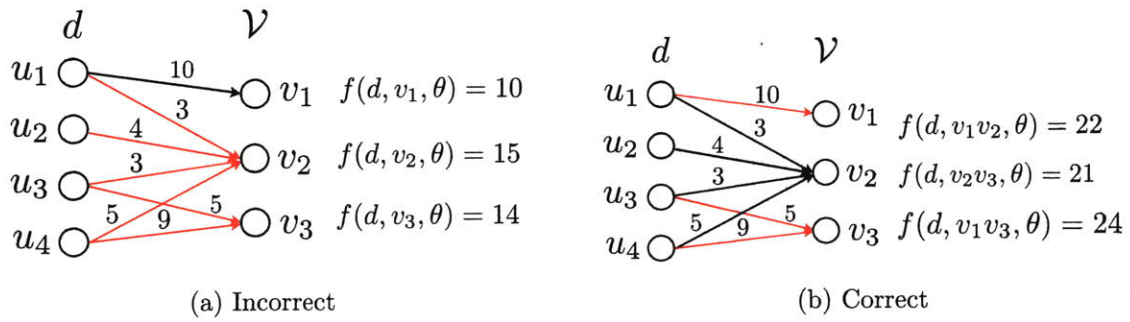
For each document d, we divide our query set $Q(d)$ into a set of positive and negative examples denoted as $Q^+(d)$ and $Q^-(d)$ respectively using a document specific query score threshold $\rho(d)$. The motivation for this was presented in the previous section 4.5.3, we are only interested in learning the sub-space of good queries and ranking them higher than the rest. Additionally, we require that each long query word $u \in d$ maps to exactly one reformulated query word $v \in q$. We specify the mapping between u and v with binary variables z_{uv}, $u \in d$, $v \in q$, so that $z_{uv} = 1$ if u is assigned to v and zero otherwise. More formally, for any d and q, the set of valid mappings is given by:

$$Z(d,q) = \left\{ z_{uv} \in \{0,1\}, u \in d, v \in q \ : \ \sum_{v' \in q} z_{uv'} = 1 \right\}$$

The translation score from d to q is then given by:

$$f(d,q;\theta) = \max_{z \in Z(d,q)} f(d,q,z;\theta) = \max_{z \in Z(d,q)} \sum_{u \in d, v \in q} \theta_{uv} z_{uv} = \sum_{u \in d} \max_{v \in q} \theta_{uv}$$

If we think of the short query words as "clusters" then each long query word is assigned to the best cluster. Note that some of the short query words in this formulation may not have any long query words assigned to them. We would never predict such reformulations but can nevertheless evaluate their score within the model. Note that once the parameters are learned, $argmax\_q$ can no longer be output by



(a) Incorrect

(b) Correct

Figure 4-7: Example demonstrating that $argmax\_q$ of length 2 can no longer be obtained by computing a score for each term separately and concatenating 2 highest scoring terms.

computing $f(d,v,\theta)$ for all the words $v \in \mathcal{V}$ separately and concatenating the highest scoring ones. The main reason is that each term in the long query can now map to at most one term in the reformulated query. Figure 4-7 presents a simple example. Given the long query d consisting of four terms, we want to output an $argmax\_q$ of length two. The learned parameter value $\theta_{u_i v_j}$ is represented on the edge connecting nodes u_i and v_j. The procedure presented in section 4.5.1 will output $argmax\_q$ consisting of terms v_2 and v_3 whereas the correct $argmax\_q$ is composed of v_1 and v_3.

Given a training set of pairs $T_n = \{(d_i, R_i, Q^+(d_i), Q^-(d_i))\}_{i=1,\ldots,n}$, the constraints that our learning algorithm needs to satisfy are:

$$\forall i,\ \forall q \in Q^+(d_i) : f(d_i,q;\theta) \geq \rho(d_i) + 1 - \xi_{i,q},\ \xi_{i,q} \geq 0$$

$$\forall i,\ \forall q \in Q^-(d_i) : f(d_i,q;\theta) \leq \rho(d_i) - 1 + \xi_{i,q},\ \xi_{i,q} \geq 0$$

$\rho(d_i)$ is a long query specific threshold since the query score distribution can vary across long queries. The objective that we need to minimize is:

$$J(\theta; S_n) = \frac{1}{2}\|\theta\|^2 + \frac{C}{n}\sum_{i=1}^{n}\left(\frac{1}{|Q^-(d_i)|}\sum_{q\in Q^-(d_i)}\xi_{i,q} + \frac{1}{|Q^+(d_i)|}\sum_{q\in Q^+(d_i)}\xi_{i,q}\right)$$

subject to the constraints that can be re-written as:

$$\begin{aligned} &\forall i,\ \forall q \in Q^+(d_i) : \xi_{i,q} \geq \rho(d_i) + 1 - \sum_{u\in d_i}\max_{v\in q}\theta_{uv},\ \xi_{i,q} \geq 0 \\ &\forall i,\ \forall q \in Q^-(d_i) : \xi_{i,q} \geq \sum_{u\in d_i}\max_{v\in q}\theta_{uv} - \rho(d_i) + 1,\ \xi_{i,q} \geq 0 \end{aligned} \tag{4.6}$$

The constraints (4.6) added by the set of good queries form a non-convex set. However, we can make it convex by fixing the mapping $z^{iq} \in Z(d_i, q)$ for each (d_i, q) where $q \in Q^+(d_i)$. The optimization constraint then becomes:

$$\forall i,\ \forall q \in Q^+(d_i) : \xi_{i,q,z^{iq}} \geq \rho(d_i) + 1 - \sum_{u\in d_i, v\in q}\theta_{uv}z^{iq}_{uv},\ \xi_{i,q} \geq 0$$

The goal is to minimize the objective jointly over $(\theta, \{z^{iq}\}_{i=1,\ldots,n})$:

$$J'(\theta, \{z^{iq}\}_{i=1,\ldots,n}; S_n) = \frac{1}{2}\|\theta\|^2 + \frac{C}{n}\sum_{i=1}^{n}\left(\frac{1}{|Q^-(d_i)|}\sum_{q\in Q^-(d_i)}\xi_{i,q} + \frac{1}{|Q^+(d_i)|}\sum_{q\in Q^+(d_i)}\xi_{i,q,z^{iq}}\right)$$

and this can be done iteratively. First, for a fixed θ, we find the mapping z^{iq} corresponding to each $(d_i, q \in Q^+(d_i))$, then apply the method above to solve for new θ, and iterate. The objective will be monotonically decreasing but we are no longer guaranteed to find the globally optimal solution.

We will not explore the proposed model enhancement in this thesis, but we believe it is an interesting new direction for future work.

4.5.5 Discussion

We have presented a novel machine learning approach to query reformulation that carries out both query reduction and translation. Our iterative learning algorithm is able to start from a small set of candidate queries, letting us automatically evaluate queries rather than relying on human judgments of document relevance with respect to the queries. The approach that we presented employs simple word-to-word mappings as a feature set. We highlight two potential problems that we noticed with our choice of the parameters.

First, our approach requires a lexical match between the pair of words seen in the training and test data. Thus the model lacks any signal for rare but potentially useful words present in the held out long query. A heuristic can be used to smooth the parameters of the model for unseen words. However, we believe that over fitting will still occur even when such heuristics are used. Perhaps it would make sense to do semantic analysis of the text in the long query to infer the most representative topics and pick the combination of query terms that cover some of those topics well. Work done by the NLP community in the area of text summarization can possibly be leveraged to augment the current approach to get some signal for rare query terms. Work by Edmundson [15] and Luhn'59 [26] studied the techniques of extracting sentences based on sentence position in the text (first paragraph, or immediately following section headings), appearance of lexical cues (*significant, hardly, impossible*), and location (first and last sentences of each paragraph [19]. It would be interesting to extend this idea to generate appropriate features that would extract discriminative query terms from a long query by treating the long query as a natural language query (for instance, the abstract as it is) rather than just an unordered collection of terms. Section 4.6 explores this approach by training a model using a combination of semantic features extracted from a learned topic model over the corpus, features inspired from text summarization literature and the standard query features.

Second, treating every pair of words as features yields a rich but very large feature set. As a result, the model performs very well by over fitting the small amount of

training data. What we need is a stronger regularization or dimensionality reduction to improve generalization. An approach similar to Latent Semantic Indexing [6] can be used which will give us a couple of advantages. Since there are many ways to express a given concept, the literal terms present in the training data may not match the ones entered by the user. LSI style model will help overcome the problem of lexical matching by representing those terms in the same latent semantic space. Use of a low rank model for dimensionality reduction will also reduce the size of the feature set which should help with generalization. Application of low rank model to query reformulation is a novel contribution of our work and we describe this approach in detail in Section 4.7.

4.6 Feature Set II: RB

In this section, we propose a **R**ule **B**ased feature set which assists in identifying key query terms from the user query. The proposed approach tries to overcome the deficiency of the **T**ranslation **M**odel (presented in section 4.5) which lacked signal for unseen/rare terms present in the input query. TM considers the long query as a bag of words and ignores the ordering of words which can be potentially useful in isolating prominent terms from the long query. Here, the problem setting is similar to the one presented in Section 4.7.6. A long query can be treated as a query document entered by the user, and we will explore techniques for determining which query terms are important to keep in the reformulated query.

We propose a combination of query, document structure and semantic features. The query specific features are standard IR features such as frequency or the *idf* of a query term. The document structure features in our feature pool are inspired from the text summarization literature [30, 15, 19]. The position or placement of a word in the long query can be a good indicator of its importance for query reformulation task. The idea behind semantic features is that the long query (such as abstract) can be seen as a small selection from the complete discourse that is written on its topic. A word that appears in a sentence can either serve a syntactic function or provide semantic content [17]. This set of features allows us to interpret the text at the semantic level in order to isolate words that are indicative of the content that ties the long query with the relevant documents at a semantic level. As such, we want to distinguish topically related words from the long query.

To provide an idea of the range of features that can be used, we now briefly describe the style of features that we incorporated in the RB model.

4.6.1 Query Features

These are the typical query term level statistics derived from the document corpus. d below refers to the long query and q is the reformulated query.

Term Frequency

$$\phi_{tf}(q,d) = \sum_{v \in q} tf(v,d)$$

where $tf(v,d)$ is the term frequency of term v in the long query d.

Mutual Information

This feature is computed similarly to the feature presented in [24]. Let T_{xy} be the number of times terms x and y occur in the corpus within a term window of 100 terms. T_x and T_y are the term frequencies of x and y and T is the total number of terms in the collection. Mutual Information score for terms x and y is given by

$$I(x,y) = \log \frac{\frac{T_{xy}}{T}}{\frac{T_x}{T}\frac{T_y}{T}} \tag{4.7}$$

The above score (4.7) is calculated for each pair of terms in the query and an average is taken over the number of pairs. This gives us the Mutual Information feature value ϕ_{mi} for a given query.

Query Length

$$\phi_{len}(q) = \begin{cases} 1 & \text{if query length is } len \\ 0 & \text{otherwise.} \end{cases}$$

We have a separate feature for queries of different lengths as retrieval effectiveness is known to degrade as the query length increases.

4.6.2 Document Structure Features

This set of features depends on the format and style of writing for a particular domain. They can be especially useful in the query-by-document scenario, where the user

provides the portion of the document as a natural language query. For our corpora, we propose a feature set inspired by work done in the area of text summarization. We present a way to automatically learn the optimal values of these features, which means they can be easily learned for different document collections.

Title

The goal of this feature is to extract discriminative words from the skeleton of the document (title, headings etc.) if that information is available. The skeleton of the document often outlines the subject matter of the document and hence the words that appear in the skeleton can be of strong significance.

$$\delta_{title}(v,d) = \begin{cases} 1 & \text{if query term } v \text{ is present in title} \\ 0 & \text{otherwise.} \end{cases}$$

$$\phi_{title}(q,d) = \sum_{v \in q} \delta_{title}(v,d)$$

Capitalization

Capitalization feature can be used to capture the importance of name entities and acronyms for appearing in shortened queries. For instance, authors often coin new or refer to established acronyms like PCA (Principal Component Analysis), SVM etc. which can be useful for finding related content.

$$\delta_{caps}(v,d) = \begin{cases} 1 & \text{if } v \text{ appeared capitalized in } d \\ 0 & \text{otherwise.} \end{cases}$$

$$\phi_{caps}(q,d) = \sum_{v \in q} \delta_{caps}(v,d)$$

Note that $\delta_{caps}(v, d)$ is 0 if v appeared capitalized only in the title or as the first word of a sentence.

Sentence Position

Just as the first and last sentences of the paragraph are considered useful while creating summaries of text [15], we extend this idea to locate positions within a sentence where informative words are likely to appear. Only punctuation appearing within the sentence is removed before calculating word positions.

$$\delta_{pos}(v, d) = \begin{cases} 1 & \text{if } v \text{ at position } pos \text{ in sentence in d} \\ 0 & \text{otherwise.} \end{cases}$$

$$\phi_{pos}(q, d) = \sum_{v \in q} \delta_{pos}(v, d)$$

There can be multiple features for different values of *pos*. For instance, a feature can be defined for *pos* 1 (first word of sentence), -1 (last word of sentence), -2 (second last word) etc. We included 6 features ϕ_1 to ϕ_6 corresponding to *pos* = 1, 2, 3, -1, -2, -3.

Phrase Cues

Hovy and Lin [19] show that sentences following certain phrases can be good indicators of important content. This feature tries to isolate candidate words from sentences containing important content that can be useful for query reformulation.

$$\delta_{phrase}(v, d) = \begin{cases} 1 & \text{if } v \text{ followed } phrase \text{ in sentence in d} \\ 0 & \text{otherwise.} \end{cases}$$

$$\phi_{phrase}(q, d) = \sum_{v \in q} \delta_{phrase}(v, d)$$

We manually compiled the list of 200 cue phrases from our document corpus. Some *phrases* that we included are "this paper", "we show that", "we propose", "we present" etc. As an example, consider a couple of sentences from an abstract of a paper:

> *This paper presents the first Rademacher complexity-based... We present the first margin bounds for kernel-based...*

The rare term "Rademacher" and "margin" were found to be present repeatedly in the top reformulated query set for this abstract.

4.6.3 Semantic Features

We treat a long query as a prior knowledge of relevant documents that the user is interested in retrieving. As such, we want to find a way of predicting what terms represent the essence of the topics present in this set of documents. However, the words used by searchers are often not the same as those by which the information they seek has been indexed. The problem is exacerbated by the fact that the user queries are extremely short, which makes it hard to infer various topical aspects from the query. Therefore, topic models do not directly address the "vocabulary mismatch" problem but they do capture variability of terms associated with any particular topic. If the system is provided with a long portion of text containing the vocabulary that the user finds relevant, we believe that the topic models learned over the document corpus can be useful in providing us with the main topics that the user is interested in exploring more.

A specific implementation of learning a topic model over a document corpus is Latent Dirichlet Allocation [7]. LDA is a generative probabilistic model of a corpus. The generative process for each document d involves picking a multinomial distribution θ_d over topics from a Dirichlet distribution. Then for each word, a topic t is chosen from this topic distribution. Finally, the word w is generated from topic-

specific multinomial $p(w|t)$. Once the model is learned, it can be used to infer the topic distribution $p(t|d)$ for a given document d.

These are the list of features we derive from the LDA model:

Topic Coverage

This feature is used to capture how well a reformulated query covers the main topical aspects present in the long query. The topic score for a query is given by:

$$\phi_{tscore}(q,d) = \sum_{t \in T} p(t|d) \sum_{v \in q} p(v|t) \tag{4.8}$$

where T is the set of top M inferred topics that have $p(t|d)$ probability value above a certain threshold. We expect good query reformulations to cover the main ideas presented in the long query well.

Topic Consistency

Long queries are typically multi-faceted. The abstract of a paper is usually a novel way of bringing together a subset of existing topics from a broader collection. In principle, this would suggest using diverse query words to cover the key topics. However, each reference of a paper typically provides more knowledge about a few specific topics out of the ones present in the abstract. Since, the search engine only retrieves documents that contain all the query words, it is important to ensure that query words are topically consistent. So we construct the following feature as a way of measuring topic consistency between query terms.

$$\phi_{tcons}(q,d) = \begin{cases} 1 & \text{if } \exists t \in T \text{ s.t. } p(v|t) > \tau \; \forall v \in q \\ 0 & \text{otherwise.} \end{cases}$$

τ is a threshold that can be tuned using a validation set.

4.6.4 Evaluation

We evaluated our model using *Data Set II.* Given the input set of features described in the earlier sections, we used the learning framework presented in Section 4.2.1 to rank the top 1000 queries for each long query.

We trained the LDA model over the entire corpus with 8927 documents using MALLET [27] toolkit. Table (4.16) shows first few words from a few topics learned by LDA model with number of topics set to 20.

Table 4.16: Example Topics learned by LDA model

speech	*reinforcement learning*	*stimuli*	*optimization*
word	learning	neurons	problem
model	state	cells	convex
language	policy	spike	function
speech	action	visual	linear
topic	methods	response	set
document	function	neuron	optimization
text	reinforcement	model	method
hmms	reward	stimulus	dual

The metric used to evaluate the quality of *top_q* is described in Section 4.2.3.

Parameter Settings

This section describes how various parameters and thresholds were set for the model.

Regularization Parameter (C): The regularization parameter was selected within the set {0.0001,0.001,0.01,0.1,1}. All the results henceforth were obtained with regularization parameter C set to 0.001.

Number of Topics (N): In LDA, the most important parameter is the number of topics. We experimented with $N = 20$ and 200. The performance difference was not substantial but $N = 200$ performed slightly better so our experiments were carried out with number of topics set to 200.

Topic Coverage (T): This parameter is defined in equation (4.8). We experimented with a few different values of T. Table (4.17) presents the results. It can be seen that the value of T has little effect on the results so for rest of the experiments

T was chosen to be 10.

Table 4.17: RB: Effect of Topic Coverage threshold on performance

Validation Set	
T	MAP
5	0.131
10	**0.137**
15	0.136

Topic Consistency Threshold (τ): τ is explained in equation (4.9) and results for varying τ values are presented in table (4.18). It seems to have relatively significant affect on the performance on validation set. We picked τ to be 0.001 for our final experiments.

Table 4.18: RB: Effect of Topic Consistency threshold on performance

Validation Set	
τ	MAP
0.0001	0.137
0.001	**0.141**
0.002	0.129
0.005	0.129
0.01	0.127

4.6.5 Performance Comparison

We conducted the following experiments to evaluate the performance of our model.

Feature Selection

Table (4.19) explores the importance of each of the proposed features. Each feature is removed one at a time from the training data and the results are reported on the validation set.

Topic features seem promising and so does the title. In the final comparison, we trained our model with N = 200, τ = 0.001, T = 10 and features *title, pos, tscore* and *tcons.*

Table 4.19: RB: Feature selection results on Data Set II

Feature Performance	
Feature Omitted	MAP
None	**0.132**
title	0.12
caps	0.133
tf	0.133
pos	0.127
mi	0.136
tscore,tcons	0.116
tscore	0.128
tcons	0.122
len	0.138
phrase	0.134
caps,tf,mi,len,phrase	**0.137**

Baseline Models

In these experiments, we compared our approach to the following methods:

High IDF: We generated a reformulated query of length 3 by randomly picking high *idf* terms from the long query.

TScore: We generated queries of length 2 and 3 by sorting high *idf* terms by their tscore (equation (4.8)) values and then concatenating the top few.

QP: Described in Section 4.4.

TM: We trained the word translation model presented in Section 4.5 over Data Set II. The original size of the vocabulary $|\mathcal{U}X\mathcal{V}|$ was 335080 but we removed uv pairs that appeared only in one (d, Q) pair. The resulting feature set size was 26969. The training/test data sets were filtered accordingly. The value of C was chosen from the set {0.001,0.001,0.01,0.1,1} using the validation data set. We modified the $argmax\_q$ procedure explained in Section 4.5.1 to carry out strict summarization. That is, for a new long query d, we computed a score for each term that appears in d as follows:

$$score(v) = \sum_{u \in d} \theta_{uv}, \forall v \in d \tag{4.9}$$

and concatenated the top 3 scoring terms to get $argmax\_q$.

Table 4.20 shows the performance comparison. *Oracle* and *Orig* refer to the MAP

over the best reformulated and original long queries respectively. It is evident that *RB* receives the largest number of *Gains* which is the number of long queries for which the chosen reformulated queries were better than the original query. Number of *Winners* stands for the number of test queries for which the model successfully picked the best reformulated query. 5 out of 9 *Winners* for *RB* overlapped with *TScore (Query Length 3)* and hence the gains in those cases can probably be attributed to the *tscore* feature. We looked closely at the *Winners* for *QP* and noticed that the model worked well for cases where one of the best reformulated query candidates had a dominant total *IDF* score of the query terms. However, that sometimes led to a query that was as specific as the original query resulting in a large number of *Unaffected* cases similar to *High IDF*. *TM* was evaluated on filtered test data and still performed comparable to *QP*. *TM argmax_q* is extremely efficient to compute and is one of the main contributions of the word translation model. Finally *TScore (Len 2)* has maximum number of *Losses* since we found that a reformulated query of length two is often not specific enough leading to a large recall but low precision.

Overall, it is evident that *RB* model has been able to achieve decent performance gains by selecting better reformulated queries for 74% of the test queries while reducing the number of losses using the proposed feature set.

Table 4.20: RB Baseline Comparison: Data Set II

Test Set							
Model	Oracle	Orig	MAP	# Gains	# Losses	# Unaffected	# Winners
High IDF	0.28	0.05	0.08	146	64	19	2
TScore (Len 2)	0.28	0.05	0.10	127	96	3	5
QP	0.28	0.05	0.10	131	72	12	16
TM (*top_q*)	0.28	0.05	0.11	159	51	11	10
TM (*argmax_q*)	0.28	0.05	0.11	159	50	12	10
TScore (Len 3)	0.28	0.05	0.12	159	60	3	9
RB	0.28	0.05	**0.14**	**170**	45	7	9

4.6.6 Discussion

The task of cross referencing documents from different sources or finding related content given a portion of relevant text (perhaps buried in the search results) is extremely useful. We have presented a discriminative way of approaching the problem of query by document where the system is provided with text as a long query that the user finds relevant to his information need. Our feature set provides an automatic way of isolating discriminative words by exploiting the document structure. Our research shows that inferring various topical aspects present in the query and deriving useful features from it is quite promising. The quality of topic inference is improved since the model is provided with long string of terms that are found in the index of relevant documents alleviating the problem of vocabulary mismatch. Long queries provide more context which further helps with topic inference helping us prevent query drift.

4.7 Low Rank Model: LR

In this section, we present a class of **L**ow **R**ank models that are discriminatively trained to map query-document or query-query pairs to ranking scores. The proposed model is similar in flavor to the **T**ranslation **M**odel presented in section 4.5 but requires less storage. Additionally, the low rank approach can help us deal with the problem of *synonymy* – multiple words with similar meaning – in a manner similar to Latent Semantic Indexing [13] methods.

We build on the idea proposed in the paper "Polynomial Semantic Indexing" [4]. The long query and the reformulated queries are represented as vectors in the same lower dimensional latent space. Words that function similarly on the training set become close in the latent space. In particular, words that appear in similar abstracts and appear in the top reformulated query set during training get mapped close to each other in the latent space. Thus, the model can be useful for carrying out query translation by mapping a long query to a reformulated query containing words not originally present in the long query.

The model is based on word-to-word mappings making it language independent. Thus it can be used to output a reformulated query in a target language given a long query in a different source language. One of the key challenges to LSI style models is scalability. In this work, we present a simple and effective algorithm for training the low rank model in a supervised manner which is especially suited for learning from large datasets.

4.7.1 LR Model

Given a long query $d \in \mathbb{R}^D$ and a reformulated query $q \in \mathbb{R}^D$, consider the scoring function $f_W(d, q)$ given by:

$$f_W(d, q) = \sum_{i,j=1}^{D} W_{ij} d_i q_j = d^T W q$$

where D is the size of the dictionary and the j'th dimension of a vector indicates the term frequency of j'th term, for example tf-idf weighting. However, this scoring function is exactly the same as the one used by TM (section 4.5) if the term-query vectors $d, q \in \mathbb{R}^D$ are incidence vectors i.e. the j'th dimension is set to 1 if the corresponding term is present in the query. Unfortunately, maintaining the parameter matrix W of size $D \times D$ in memory can become infeasible as the size of D grows. In order to reduce the number of parameters, TM explicitly sets certain entries to 0 by ignoring word pairs that do not appear sufficiently often in the top queries. Nevertheless, the huge number of parameters can affect the generalization ability of the model.

Developing on the idea proposed in the paper [4], we use a low rank parameterization of the above model with a new scoring function:

$$f_{UV}(d, q) = d^T(U^T V + I)q \tag{4.10}$$

where U and V are $k \times D$ matrices. The idea is similar to LSI. U and V not only induce a k-dimensional "latent concept" space but are also faster to compute and are more economical to store in memory compared to W. k is usually much smaller than D. For our problem, k was set to a value less than 200 whereas the dictionary size was 5400.

4.7.2 Pegasos for Learning LR Ranking Model

The model is trained in a supervised fashion using the preference relations as described in equation 4.2 with the margin set to one. The training data $T_n = \{(d_i, R_i, Q_i)\}_{i=1,\ldots,n}$ can be written as a set of tuples $\mathcal{T} = \{d_i, q_i^+, q_i^-\}_{i=1,\ldots,n}$ where q_i^+ and q_i^- are high and low ranked reformulated queries in Q_i respectively.

The paper [4] trains the model using stochastic gradient descent by iteratively picking a random tuple (d, q^+, q^-) from the set $\mathcal{T}$ to minimize the sum of the ranking loss over all tuples. Fixed learning rate η is provided as an input to the algorithm.

The objective is to minimize:

$$\sum_{(d,q^+,q^-)\in\mathcal{T}} \max(0, 1 - f_{UV}(d, q^+) + f_{UV}(d, q^-))$$

and the updates are given by:

$$U \leftarrow U + \eta V(q^+ - q^-)d^T, \text{ if } 1 - f_{UV}(d, q^+) + f_{UV}(d, q^-) > 0$$
$$V \leftarrow V + \eta U d(q^+ - q^-)^T, \text{ if } 1 - f_{UV}(d, q^+) + f_{UV}(d, q^-) > 0$$

The problem with the above approach is the lack of regularization over the norm of U or V. The parameter values are very sensitive to the value of the learning rate and the latter needs to be chosen carefully.

To remedy this, we propose an adaptation of Pegasos algorithm [28] used to train SVM from large datasets. Pegasos is a simple iterative algorithm which alternates between stochastic sub-gradient descent and projection steps. First, gradient update is made in the direction of sub-gradient computed over a sub-sample of training tuples. Second, updated U and V are projected onto L_2 ball of radius $\sqrt{1/\lambda}$.

The algorithm proceeds as follows. Initially, U and V are set to any matrices with Frobenius norm at most $1/\sqrt{\lambda}$. The Frobenius norm of matrix U is defined as:

$$||U||_F = \sqrt{\sum_{i=1}^{k}\sum_{j=1}^{D}|u_{ij}|^2}$$

On iteration t, a set of tuples (d, q^+ (high ranked), q^- (low ranked)), which we denote as A_t, are sampled from the entire training set $\mathcal{T}$. The approximate learning problem is to find a low rank parameter matrix that minimizes the regularized hinge loss over the sampled tuples. Hence, the goal is to minimize the approximate objective function:

$$\frac{\lambda}{2}||U||_F^2 + \frac{\lambda}{2}||V||_F^2 + \frac{1}{|A_t|}\sum_{(d,q^+,q^-)\in A_t} \max(0, 1 - f_{UV}(d, q^+) + f_{UV}(d, q^-)) \quad (4.11)$$

The above objective is not convex in the low rank parameterization. We perform iterative update for U while holding V fixed and vice versa. The learning rate is set to $\eta_t = 1/\sqrt{t}$ and a two step update is performed for each matrix. A_t^+ is defined to be the set of tuples for which non-zero loss is encountered i.e.

$$A_t^+ = \{(d, q^+, q^-)\} \text{ s.t. } 1 - f_{UV}(d, q^+) + f_{UV}(d, q^-) > 0,\ \forall (d, q^+, q^-) \in A_t \quad (4.12)$$

The first step of pegasos update for U is given by:

$$U_{t+\frac{1}{2}} \leftarrow (1 - \eta_t \lambda) U_t + \frac{\eta_t}{|A_t|} \sum_{(d,q^+,q^-) \in A_t^+} V_t (q^+ - q^-) d^T \quad (4.13)$$

U_{t+1} is obtained by projecting $U_{t+1/2}$ onto the set:

$$B = \{U : ||U||_F \leq 1/\sqrt{\lambda}\} \quad (4.14)$$

which completes the second step. The update for V can be derived similarly. First, the loss set A_{Vt}^+ is computed using updated U_{t+1}. Second, the update is calculated exactly as given by (4.12):

$$V_{t+\frac{1}{2}} \leftarrow (1 - \eta_t \lambda) V_t + \frac{\eta_t}{|A_{Vt}|} \sum_{(d,q^+,q^-) \in A_{Vt}^+} U_{t+1} d (q^+ - q^-)^T \quad (4.15)$$

followed by the projection step (4.14). Next, we compute U_{avg} and V_{avg} over the last p iterations and log the value of objective (4.11) at these parameter values as well as their performance on the validation set.

The algorithm sketch is presented in Figure 4-8. We use the validation set to determine convergence (stopping criteria).

4.7.3 Query Composed of Top Ranked Words

The exact *argmax_q* computation is very efficient for this model. We make the following observation: let a reformulated query $q \in \mathbb{R}^D$ be represented by a sparse

Inputs:
p, λ, $\mathcal{T} = \{d_i, q_i^+, q_i^-\}_{i=1,\ldots,n}$

Initialize:
Choose $U_1, V_1 \in B$ where B is defined in (4.14)

Algorithm: For $t = 1, 2, \ldots$:

1. Set $\eta_t = \frac{1}{\sqrt{t}}$
2. Choose $A_t \in \mathcal{T}$
3. Set A_t^+ as per (4.12)
4. Set $U_{t+\frac{1}{2}}$ as given by (4.13)
5. Set $U_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{||U_{t+\frac{1}{2}}||_F}\right\} U_{t+\frac{1}{2}}$
6. Set $V_{t+\frac{1}{2}}$ as given by (4.15)
7. Set $V_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{||V_{t+\frac{1}{2}}||_F}\right\} V_{t+\frac{1}{2}}$
8. Compute $U_{avg} = \sum_{l=t-p+1}^{t+1} U_l$ and V_{avg}. Log the value of objective over A_t and the performance on validation set.

Figure 4-8: Pegasos Algorithm for Training the LR Model

vector $\bar{q} = \{\bar{q}_1, \bar{q}_2, \ldots, \bar{q}_{|q|}\}$ which stores the indices in q that are set to 1. $|q|$ represents the length of the query (number of words in q). The low rank query score is given by:

$$f_{UV}(d, q) = d^T(U^T V + I)q = \sum_{i=1}^{|\bar{q}|} [d^T(U^T V + I)]_{\bar{q}_i} \tag{4.16}$$

That is, we just sum up the entries of vector $d^T(U^T V + I) \in \mathbb{R}^D$ corresponding to the words that appear in q.

To output the query composed of top ranked words, we compute $f_{UV}(d, v)$ for each word $v \in d$ in the long query separately. Next, we concatenate the top scoring 3 words to output $argmax\_q$ of length 3. The procedure is the same as first computing the vector $d^T(U^T V + I)$ and concatenating terms corresponding to indices with the

3 largest values.

4.7.4 Evaluation

We conducted experiments on Data Set II (described in Section 4.3.2) after dividing it into training/test/validation. Each query-term vector stored binary incidence values for the presence/absence of the corresponding words. During each iteration, a set of random tuples $\{(d_i, q_i^+, q_i^-)\}_{i=1...n}$ (where $q_i^+, q_i^- \in Q_i$) are chosen for 20% of the long queries. For each tuple, q_i^+ is picked randomly from top l% ranked queries in set Q_i and q_i^- from the bottom $(100 - l)$%. Thus, for each long query, the ranking constraints are imposed between the top query set comprising of top l% reformulated queries and the remaining queries. We evaluated MAP for the queries appearing at l% position from the top across the validation set to get an idea for a sensible division of queries into top and bottom query set. l can be considered as an input parameter

Table 4.21: Top and Bottom Query Set Cut-off with Oracle Score of 0.29

l	**MAP**
10	0.15
5	0.178
3	0.20
2	0.214

to our model and we analyze its effect on performance in the next section.

In order to help determine convergence, we kept track of the value of the approximate objective (4.11) and plotted it after each iteration of the algorithm at U_{avg} and V_{avg}. We measured MAP value[2] over the top ranked queries *top_q*, as explained in Section 4.2.3, on the validation set which was composed of 1000 queries for each long query. This value was plotted at every p'th iteration using U_{avg} and V_{avg} to help determine convergence and analyze performance.

We trained a number of low rank models with different input parameter settings to compare their generalization performance. p was set to 20 for all the experiments. The

[2]Defined in Section 4.1

next few sections analyze the impact of the input parameters k, l, λ on performance which was assessed using the validation set as explained above.

Objective Value

Figure 4-9 is a typical graph which shows fluctuations in the approximate objective function as the algorithm progresses by taking gradient steps with respect to the sub-samples. The blue line represents the value of the objective function calculated on the sampled training tuples A_t. We can see that the overall objective value decreases with the increase in the number of iterations. The red line plots the approximate objective value obtained on a random sample of tuples chosen from the validation set. It is evident that the objective value calculated on the validation set decreases initially. However, it starts to increase giving us an indication of over-fitting the training data as the learning procedure continues. To get a smoother curve, we plot an average of the individual objective values obtained from past 10 iterations for each point.

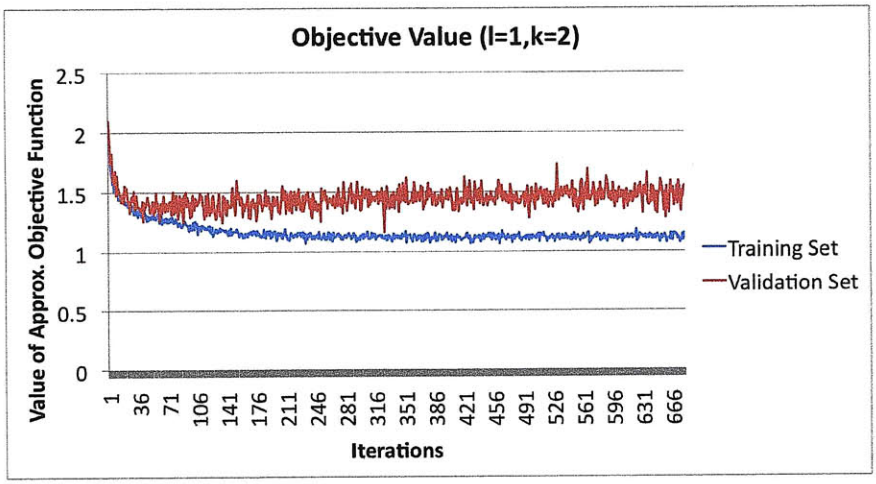
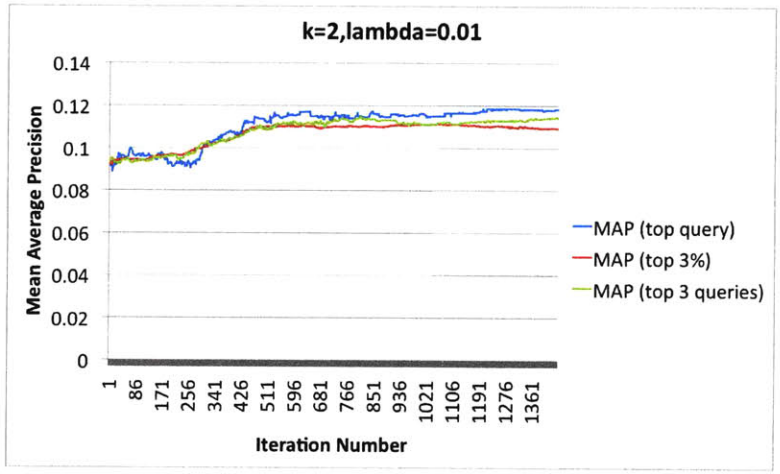


Figure 4-9: Approx. objective values computed at step (8) of training algorithm (figure 4-8) at various iterations.

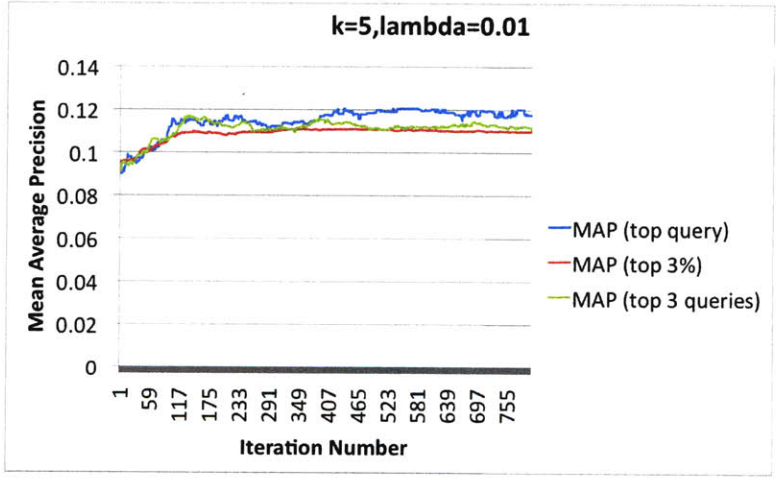
Change in Evaluation Metric

This section explores the change in the criteria used for evaluating the performance of the model. Instead of measuring the score of top ranked query, we computed the average score of top 3% queries and 3 queries for each long query. Since the ranking

constraints are imposed between top and bottom query sets, the intuition was that the model might not be able to rank the query with the highest score at the top. So we computed the average over top few queries to see whether the model is able to rank good queries at the top positions. l was set to 3 for the plots shown in Figure (4-10).



(a)



(b)

Figure 4-10: MAP value on validation set according to 3 different evaluation metrics computed at each iteration of training algorithm (Figure 4-8 step 8).

The blue line represents MAP over single top ranked queries by the model. It is surprising to note that the score of the top ranked query is higher than the average over a top few. Typically each long query has more than one best scoring query. It is possible that by forcing ranking constraints between top few queries and the

remaining, the learned model is able to rank one of the good queries at the top position but is unable to do so consistently for a top few positions. We continue to use 4.2.3 as a metric for the rest of our experiments.

Varying value of top query set size: l

This section analyzes the change in size of the top and bottom query set for enforcing ranking constraints. Evaluation results for varying l are presented in Figure 4-11.

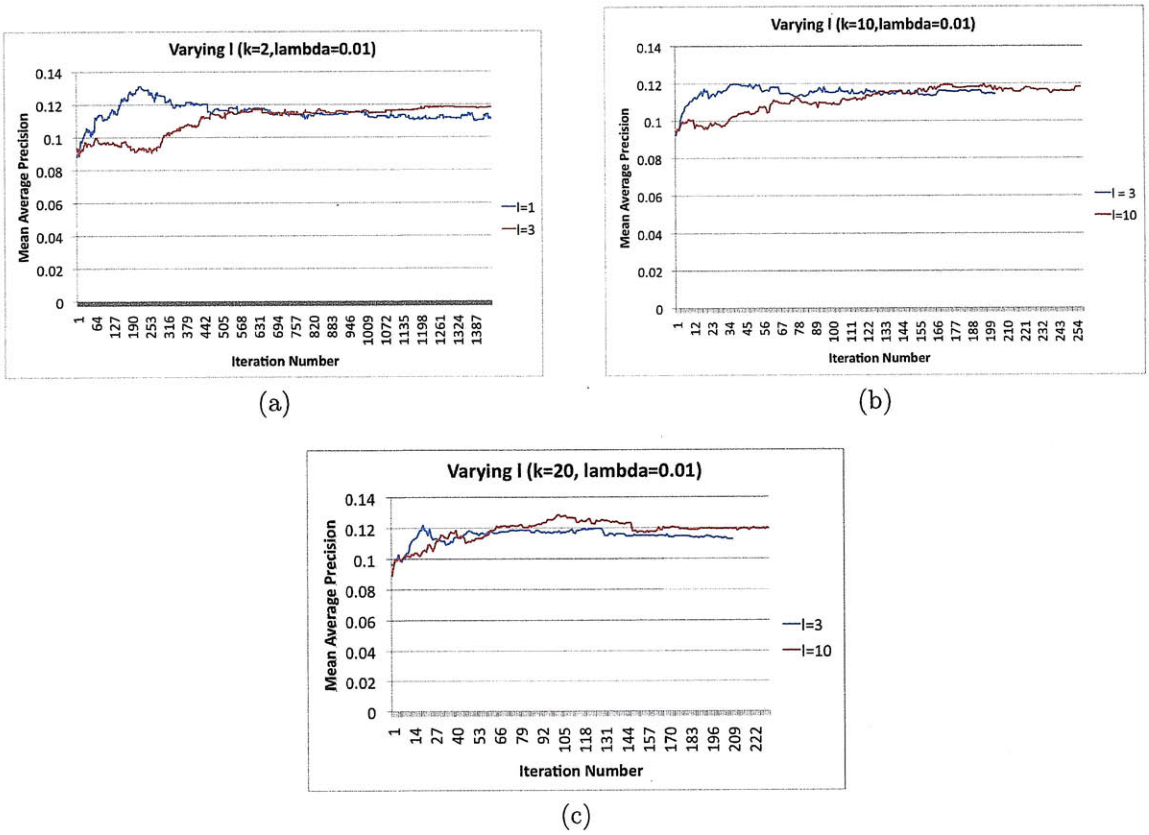


Figure 4-11: Effect on performance due to change in top query set size for imposing ranking constraints. MAP is computed at various iterations of training algorithm over validation set.

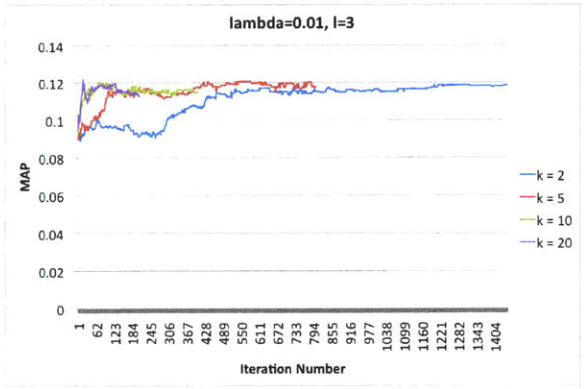
It seems using a small set of top ranked queries to enforce ranking constraints leads to a quick improvement in the performance of the model. Since the problem is most constrained for $k = 2$, the model learns to quickly focus on relevant features once the set of most useful constraints are imposed. However, it seems having additional ranking constraints (since the training data set is small) helps in better generalization

if the model is trained for a longer period of time. One reason for this might be that we include, in each iteration, a random subset of bottom 100-l% queries as "negatives" (q^-). This random subset will include pairs where "positive" (q^+) is in top 1% while the "negative" lies in top 2% queries. In other words, the method will try to distinguish between top queries but fails to do so. One way to avoid this would be to select the "negative" queries from bottom 90% (or something similar) regardless of l. This idea can be explored in the future experiments.

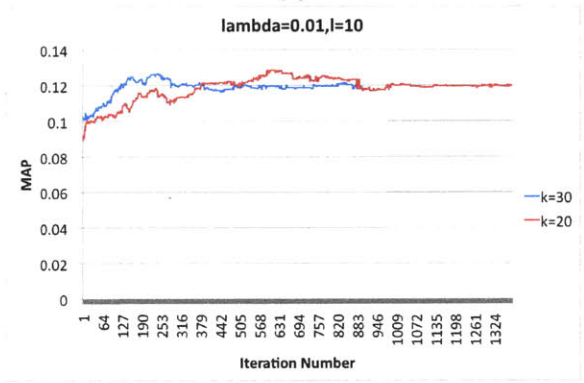
Varying latent space dimension: k

The choice of latent space dimension is critical since it should be large enough to provide enough flexibility in capturing the real structure of the data but small enough to ignore noise, redundancy and to keep the memory footprint small. Figure (4-12) shows the performance of models trained with different latent space dimensions. Since the increase in k makes each iteration of the algorithm more computationally intensive, $k = 20$ has fewest data points as the different models were trained for roughly the same amount of time. It is evident that as k is increased, it takes fewer iterations for the model to achieve its peak performance. However, the training time until convergence is still comparable.

It makes sense that the updates are faster when k is higher since the effective learning rate is higher. For example, consider $k = 2$ vs $k = 20$. In the latter case, there are 10 times more features that get updated at the same rate. Even if the actual relationship required only 2 features, the solution will get there 10 times faster since all the feature updates (10 pairs of 2) would point in roughly the same direction. Furthermore, fewer dimensions lead to a more constrained optimization problem. Since we bound the matrix norm, larger number of dimensions provide more flexibility while forcing it to focus on the most relevant features.



(a)



(b)

Figure 4-12: Effect of latent space dimension k on model performance during training.

4.7.5 Performance Comparison

This section contains a performance comparison of the Low Rank model with the previous approaches. QP is described in section 4.4 and the RB model is trained using the feature set listed in section 4.6. TM refers to the translation model trained by incorporating word to word mappings as features fully described in Section 4.5. TM's implementation details specific to Data Set II can be found in Section 4.6.5. We will refer to the queries in the validation set that do not have corresponding word-mappings in the training set as *filtered* queries.

Top 1000 Test Set

All the models are used to rank the top 1000 queries in the test set. *Random* just picks a random query from the top 1000 reformulations. It is evident that LR receives

more number of *Gains* than the TM or QP models. *# Gains* are the number of long queries for which the chosen reformulated queries were better than the original query. 49 reformulated queries output by LR model belonged to the filtered query set. 16 out of 49 of those cases had worse query scores than the corresponding TM model. *Winners* stands for the number of long queries for which the model successfully picked the best reformulated query. 1 *Winner* was common between the TM and LR models and 2 of the *Winners* for LR belonged to the filtered query set. It seems that LR helps handle the data sparsity problem that TM suffers from. The *TM* and LR models shared 5 *Unaffected* cases where the reformulated query did not offer any gain or loss over the original long query. We also looked at the *Losses* for both the LR and RB models and noticed that some of them are a result of the models not considering interactions between the chosen query words. A few examples of such top queries are: "receptive cells cell", "synapses cortex cortical", "policy hierarchy hierarchical" and "vocabularies vocabulary candidate".

As pointed out in the performance analysis in section 4.6.5, QP model's *Winners* were queries with dominant total *idf* score (feature listed in section 4.4) compared to rest of the reformulated queries. We believe that this feature might have proved especially useful since the search engine uses *tf-idf* as a scoring mechanism (given by equation 3.1).

Table 4.22: LR Baseline Comparison: Data Set II (Test Top 1000)

Test Set Top 1000						
Model	**Oracle**	**MAP**	# Gains	# Losses	# Unaffected	# Winners
Random	0.285	0.087	-	-	-	-
QP	0.285	0.110	144	59	11	17
TM	0.285	0.109	159	51	11	10
LR	0.285	0.121	172	39	11	9
RB	0.285	**0.149**	178	36	6	11

Test All

Instead of ranking just the top 1000 queries, models TM, QP and RB, rank all the queries in the test set and output the topmost one. *High IDF* concatenates 3 randomly

chosen high *idf* words from the long query and outputs that for each long query. LR computes $argmax\_q$ efficiently as explained in section 4.7.3. The results are quite similar to the ones presented in the previous section.

Table 4.23: LR Baseline Comparison: Data Set II (Test All)

Test Set		
Model	**Oracle**	**MAP**
High IDF	0.285	0.085
QP	0.285	0.101
TM	0.285	0.110
LR	0.285	0.112
RB	0.285	**0.137**

The LSI style approach provides a few advantages. TM has the problem of requiring a lexical match between the pair of words seen in the training and test data. It also yields a huge feature set which has the potential of performing very well by overfitting small amount of training data. Since there are many ways to express a given concept, the literal terms present in the training data may not match the ones entered by the user. The LSI style model helps address the problem of lexical matching by representing those terms in the same latent semantic space. However, neither model offers any signal for unseen words. The next section provides one way of extending the LR model to remedy this.

4.7.6 Model Extension - QBD

In this section, we propose LR model enhancement which enables us to enrich the model with additional features specific to the task at hand. Consider the **Q**uery **B**y **D**ocument scenario where the user provides a portion of text from a document or the entire document as a long query. We can incorporate additional features to provide signal for very low frequency, but often highly informative words that LR will otherwise have poor or no signal for. We represent this feature set with ϕ. The new scoring function can be written as:

$$f_{UV\mathbf{w}}(d,q) = f_{UV}(d,q) + \phi(d,q)\mathbf{w} \tag{4.17}$$

The updated objective becomes:

$$\frac{\lambda}{2}||\mathbf{w}||^2 + \frac{\lambda}{2}||U||_F^2 + \frac{\lambda}{2}||V||_F^2 + L(A_t, U, V, \mathbf{w}) \tag{4.18}$$

$$L(A_t, U, V, \mathbf{w}) = \frac{1}{|A_t|} \sum_{(d,q^+,q^-)\in A_t} \max(0, 1 - f_{UV\mathbf{w}}(d, q^+) + f_{UV\mathbf{w}}(d, q^-))$$

The stochastic gradient update for U and V stay the same but for $\mathbf{w}$, we first compute $A^+_{\mathbf{w}t}$ (similar to 4.12) using updated U_{t+1} and V_{t+1}. The update can be written as:

$$\mathbf{w}_{t+\frac{1}{2}} \leftarrow (1 - \eta_t\lambda)\mathbf{w}_t + \frac{\eta_t}{|A_{\mathbf{w}t}|} \sum_{(d,q^+,q^-)\in A^+_{\mathbf{w}t}} \phi(d, q^+) - \phi(d, q^-) \tag{4.19}$$

$\mathbf{w}_{t+1}$ is computed by projecting $\mathbf{w}_t$ onto the set B. The updated algorithm is provided in Figure 4-13.

Performance Comparison

For our experiments, we used the feature set from Section 4.6 as ϕ which is very applicable to QBD. Table 4.24 presents performance comparison results for the enhanced model.

Table 4.24: QBD Baseline Comparison: Data Set II

Test Set		
Model	**Oracle**	**MAP**
Random	0.28	0.08
QP	0.28	0.10
TM	0.28	0.11
LR	0.28	0.11
RB	0.28	0.14
QBD	0.28	**0.15**

The performance is better than the RB or LR model. We believe that the LR model can be an integral component of a query reformulation system but not necessarily the whole system. It can be augmented with additional features specific to a particular task to achieve further gains. We have presented a natural way of ex-

Inputs:
p, λ, $\mathcal{T} = \{d_i, q_i^+, q_i^-\}_{i=1,\ldots,n}$

Initialize:
Choose $\mathbf{w}_1, U_1, V_1 \in B$ where B is defined in (4.14)

Algorithm: For $t = 1, 2, \ldots$:

1. Choose $A_t \in T_n$
2. Set A_t^+ as per (4.12)
3. Set $\eta_t = \frac{1}{\sqrt{t}}$
4. Set $U_{t+\frac{1}{2}}$ as given by (4.13)
5. Set $U_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{||U_{t+\frac{1}{2}}||_F}\right\} U_{t+\frac{1}{2}}$
6. Set $V_{t+\frac{1}{2}}$ as given by (4.15)
7. Set $V_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{||V_{t+\frac{1}{2}}||_F}\right\} V_{t+\frac{1}{2}}$
8. Set $\mathbf{w}_{t+\frac{1}{2}}$ as given by (4.19)
9. Set $\mathbf{w}_{t+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{||\mathbf{w}_{t+\frac{1}{2}}||_1}\right\} \mathbf{w}_{t+\frac{1}{2}}$
10. Compute $U_{avg} = \sum_{l=t-p+1}^{t+1} U_l$, V_{avg} and $\mathbf{w}_{avg}$. Log the value of objective (4.18) over A_t and performance on validation set.

Figure 4-13: Pegasos Algorithm for Training the Extended LR Model

tending the model but there are several ways of accomplishing that. For instance, one can use an ensemble of learners to create a more robust ranking function. Fair exploration of such model enhancements is left as future work.

4.7.7 Discussion

One of the main contributions of this section is a completely new approach to query reformulation. We have described and presented a simple supervised method based on *words* only which can be regarded as a potential component of a query reformulation system. Low rank approximation helps with generalization and saves memory. The query representation by LR model is very economical. Our empirical results indicate that a small value of the latent space dimension k can probably suffice in achieving reasonable performance.

We also presented a novel way of training the low rank model. Since the run time does not depend directly on the size of the training set, the resulting algorithm is especially suited for learning from large data sets [28]. This can be especially important for learning from large scale web search logs. The presented model is fast to train and requires a smaller amount of memory to store the parameters compared to the Translation Model (section 4.5). The algorithm provides an additional advantage of not requiring to search for a good value for the learning rate, which can be a significantly more expensive computation. A predefined schedule η_t seems to work quite well in practice.

Another contribution of our work is the efficient way of outputting the top ranked query $argmax\_q$ that does not require an exhaustive computation of ranking all possible query reformulations. Efficiency is a key challenge since enumerating and evaluating all possible reformulated queries is not feasible in practice. Previous approaches provide heuristics to reduce the size of the search space, for instance, by only considering reformulations that differ from the original query in one term. Given a new long query, our algorithm carries out the computation of the top ranked query in an efficient manner. However, there is a drawback of using such a simple $argmax\_q$ procedure. The model doesn't capture relationships between the query terms present

in the reformulated query. This results in choosing reformulations that contain redundant terms. We presented a few examples in the analysis presented in section 4.7.5.

One shortcoming of the presented model is that it does not provide any signal for the new unseen words. But various extensions of the low rank model are possible. We saw in section 4.7.6 that the model can be augmented with other features specific to the task to further improve results.

The presented method does not depend on language and can be useful for learning cross-lingual query translations as well. One can explore a class of nonlinear (polynomial) models that can capture higher order relationships between words like the one presented in [4]. We can also experiment with storing actual *tf-idf* values instead of just binary incidence values in the term-query vectors to achieve further gains. Such extensions can be investigated in future work.

Chapter 5

Conclusion

In this thesis, we have addressed the problem of reformulating long search queries into more concise and effective queries. Recent user studies [2] have shown that harder search tasks tend to involve longer queries consisting of 5-7 terms. However, most search engines do not handle long queries well since they often include extraneous terms. Our results indicate that replacing the original long query with a smaller set of appropriate terms greatly improves query effectiveness.

We have presented three discriminatively trained models that naturally provide different trade-offs in terms of performance gains, scalability, computation and storage requirements. The translation (TM) and low rank (LR) models presented in sections 4.5 and 4.7, respectively, are based on features over *words* only. The TM model is simple yet powerful enough to achieve reasonable performance for larger corpuses. However, it may not generalize well as the size of the vocabulary grows. The LR model provides a low-rank approximation of TM which is especially suited for corpus with larger vocabularies. However, it requires as input a few carefully picked parameters, such as the dimension of the latent space. The Rule Based model from section 4.6 shows that it is possible to achieve significant performance gains by incorporating document structure and the style of writing. The flexibility in the choice of models is valuable when designing a system, where the familiarity with the corpus, memory and processing limitations may influence the choice of the model.

Several unanswered questions remain. The search engine that we employed uses a

very simple indexing and *tf-idf* based scoring (equation 3.1) scheme. A natural question then is to see how well the proposed models perform for modern search engines that employ more complex retrieval and ranking functions. We need to evaluate our approach on long queries in the web environment. The TM and LR models suffer from a lack of signal for unseen words. The RB model addresses this with the addition of features which exploit document structure and semantic features that are extracted from the topic model learned over the corpus. A more general approach could be to impose a regularization on the low rank model in a manner that fills in feature values for unseen words. For instance, one can append the following quantity:

$$\lambda \sum_{i,j} w_{ij} ||V_{\bullet i} - V_{\bullet j}||^2$$

to the objective function 4.11. The weight w_{ij} between a pair of words measures how similar the two words are, such as their mutual information score (given by equation 4.7). $V_{\bullet i}$ denotes the i'th column of matrix V corresponding to the i'th word in the dictionary. The learning procedure will steer the parameters so that the strongly coupled words will have similar representations in the latent space.

The learning formulations presented are purely discriminative. A complementary approach would be to train a generative model over documents guided by words that are good for query reformulation. The idea would be to generate the words in the document from a restricted set of words that best explain the entire document. The choice of words can be viewed as a compression of the original document. Such a generative model can have various interesting applications. For instance, it could be used to extract keywords from web pages for advertisement targeting, automatically create e-mail labels, annotate a corpus (such as blogs) with keywords to provide some structure making the corpus easier to browse. While promising, there are various challenges associated with this approach. It is unclear how well the compressed document works as a query as the generative process is not influenced by the search engine. For example, most search engines require all the search terms to be present in the document for it to be retrieved.

The goal of our research is to reformulate user descriptions to effective search engine terms, bridging vocabularies. This leads to several interesting new problems in a real system. Given the user query logs, how can we construct novice and expert query pairs? Does it suffice to analyze the query reformulation patterns within the user search sessions [32]? Or do we need to be able to distinguish novice or expert users based on their search behaviors [2]? Once we have such query pairs as training data, variants of the models suggested in this thesis can be applied to carry out query reformulation. However, it may be prohibitively expensive to translate every user query. One can predict query performance and only reformulate poorly-performing queries [11]. Future work in these directions will bring us closer to having a unified query reformulation system which would substitute, drop and append query terms typed in by novice searchers.

Bibliography

[1] Anne Aula. Query formulation in web information search. In *Proc. IADIS International Conference WWW/Internet*, volume I, pages 403–410. IADIS Press, 2003.

[2] Anne Aula, Rehan M. Khan, and Zhiwei Guan. How does search behavior change as search becomes more difficult? In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 35–44, New York, NY, USA, 2010. ACM.

[3] Leif Azzopardi. Query side evaluation. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 556–563, New York, NY, USA, 2009. ACM.

[4] Bing Bai, Jason Weston, David Grangier, Ronan Collobert, Kunihiko Sadamasa, Yanjun Qi, Corinna Cortes, and Mehryar Mohri. Polynomial semantic indexing. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems 22*, pages 64–72. 2009.

[5] Michael Bendersky and W. Bruce Croft. Discovering key concepts in verbose queries. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 491–498, New York, NY, USA, 2008. ACM.

[6] Michael W. Berry, Susan T. Dumais, and Gavin W. O'Brien. Using linear algebra for intelligent information retrieval. *SIAM Rev.*, 37(4):573–595, 1995.

[7] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, 2003.

[8] Chris Buckley, G. Salton, J. , and A. Singhal. Automatic query expansion using smart: Trec 3. In *In Proceedings of The third Text REtrieval Conference (TREC-3)*, pages 69–80.

[9] Guihong Cao, Jian-Yun Nie, Jianfeng Gao, and Stephen Robertson. Selecting good expansion terms for pseudo-relevance feedback. In *SIGIR '08: Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 243–250, New York, NY, USA, 2008. ACM.

[10] Steve Cronen-Townsend, Yun Zhou, and Bruce Croft. A framework for selective query expansion. In *CIKM '04: Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 236–237, New York, NY, USA, 2004. ACM.

[11] Steve Cronen-Townsend, Yun Zhou, and W. Bruce Croft. Predicting query performance. In *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 299–306, New York, NY, USA, 2002. ACM.

[12] Hang Cui, Ji-Rong Wen, and Jian-Yun Nie. Query expansion by mining user logs. *IEEE Trans. on Knowl. and Data Eng.*, 15(4):829–839, 2003. Member-Ma, Wei-Ying.

[13] Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

[14] Geoffrey B. Duggan and Stephen J. Payne. Knowledge in the head and on the web: using topic expertise to aid search. In *CHI '08: Proceeding of the twenty-*

sixth annual SIGCHI conference on Human factors in computing systems, pages 39–48, New York, NY, USA, 2008. ACM.

[15] H. P. Edmundson. New methods in automatic extracting. 1969.

[16] Bruno M. Fonseca, Paulo Golgher, Bruno Pôssas, Berthier Ribeiro-Neto, and Nivio Ziviani. Concept-based interactive query expansion. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 696–703, New York, NY, USA, 2005. ACM.

[17] Thomas L. Griffiths, Mark Steyvers, David M. Blei, and Joshua B. Tenenbaum. Integrating topics and syntax. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, volume 17, pages 537–544, Cambridge, MA, 2005. MIT Press.

[18] Yunsong Guo and Carla Gomes. Ranking structured documents: a large margin based approach for patent prior art search. In *IJCAI'09: Proceedings of the 21st international jont conference on Artifical intelligence*, pages 1058–1064, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[19] E. Hovy and C. Lin. Automated text summarization in summarist. In *Advances in Automatic Text Summarization*, pages 81–94. MIT Press, 1999.

[20] Lngrid Hsieh-yee. Effects of search experience and subject knowledge on the search tactics of novice and experienced searchers. *Journal of the American Society for Information Science*, 44:161–174, 1993.

[21] Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM.

[22] Thorsten Joachims. Training linear svms in linear time. In *KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, New York, NY, USA, 2006. ACM.

[23] Thorsten Joachims, Thomas Finley, and Chun-Nam Yu. Cutting-plane training of structural svms. *J. Mach. Learn. Res.*, 77:27–59, 2009.

[24] Giridhar Kumaran and James Allan. A case for shorter queries, and helping users create them. In *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference*. ACL, 2007.

[25] Giridhar Kumaran and Vitor R. Carvalho. Reducing long queries using query quality predictors. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 564–571, New York, NY, USA, 2009. ACM.

[26] H. P. Luhn. The automatic creation of literature abstracts. In *IBM Journal of Research and Development*, pages 159–165, 1958.

[27] Andrew Kachites McCallum. Mallet: A machine learning for language toolkit. 2002. http://mallet.cs.umass.edu.

[28] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 807–814, New York, NY, USA, 2007. ACM.

[29] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. *SIGIR Forum*, 33(1):6–12, 1999.

[30] Simone Teufel and Marc Moens. Summarizing scientific articles: experiments with relevance and rhetorical status. *Comput. Linguist.*, 28(4):409–445, 2002.

[31] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *J. Mach. Learn. Res.*, 6:1453–1484, 2005.

[32] Xuanhui Wang and ChengXiang Zhai. Mining term association patterns from search logs for effective query reformulation. In *CIKM '08: Proceeding of the 17th ACM conference on Information and knowledge management*, pages 479–488, New York, NY, USA, 2008. ACM.

[33] Ryen W. White, Susan T. Dumais, and Jaime Teevan. Characterizing the influence of domain expertise on web search behavior. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 132–141, New York, NY, USA, 2009. ACM.

[34] Yang Xu, Gareth J.F. Jones, and Bin Wang. Query dependent pseudo-relevance feedback based on wikipedia. In *SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 59–66, New York, NY, USA, 2009. ACM.

[35] Yin Yang, Nilesh Bansal, Wisam Dakka, Panagiotis Ipeirotis, Nick Koudas, and Dimitris Papadias. Query by document. In *WSDM '09: Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 34–43, New York, NY, USA, 2009. ACM.

[36] Wen-tau Yih, Joshua Goodman, and Vitor R. Carvalho. Finding advertising keywords on web pages. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 213–222, New York, NY, USA, 2006. ACM.

[37] Liron Zighelnic and Oren Kurland. 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*, pages 825–826, New York, NY, USA, 2008. ACM.