ISBN 978-82-471-3264-7 (printed ver.)
ISBN 978-82-471-3265-4 (electronic ver.)
ISSN 1503-8181

Nattiya Kanhabua
Doctoral Thesis

Doctoral theses at NTNU, 2012:5

Nattiya Kanhabua

Time-aware Approaches to Information Retrieval

NTNU
Norwegian University of
Science and Technology
Thesis for the degree of
Philosophiae Doctor
Faculty of Information Technology, Mathematics and
Electrical Engineering
Department of Computer and Information Science

NTNU
Norwegian University of
Science and Technology

Doctoral theses at NTNU, 2012:5

NTNU

NTNU
Norwegian University of
Science and Technology

Nattiya Kanhabua

Time-aware Approaches to Information Retrieval

Thesis for the degree of Philosophiae Doctor

Trondheim, February 2012

Norwegian University of
Science and Technology
Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

Norwegian University of
Science and Technology

NTNU
Norwegian University of Science and Technology

Thesis for the degree of Philosophiae Doctor

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

ISBN 978-82-471-3264-7 (printed ver.)
ISBN 978-82-471-3265-4 (electronic ver.)
ISSN 1503-8181

Doctoral Theses at NTNU, 2012:5

Printed by Tapir Uttrykk

To my family

Abstract

In this thesis, we address major challenges in searching *temporal document collections*. In such collections, documents are created and/or edited over time. Examples of temporal document collections are web archives, news archives, blogs, personal emails and enterprise documents. Unfortunately, traditional IR approaches based on term-matching only can give unsatisfactory results when searching temporal document collections. The reason for this is twofold: the contents of documents are strongly time-dependent, i.e., documents are about events happened at particular time periods, and a query representing an information need can be time-dependent as well, i.e., a *temporal query*.

Our contributions in this thesis are different time-aware approaches within three topics in IR: content analysis, query analysis, and retrieval and ranking models. In particular, we aim at improving the retrieval effectiveness by 1) analyzing the contents of temporal document collections, 2) performing an analysis of temporal queries, and 3) explicitly modeling the time dimension into retrieval and ranking.

Leveraging the time dimension in ranking can improve the retrieval effectiveness if information about the creation or publication time of documents is available. In this thesis, we analyze the contents of documents in order to determine the time of non-timestamped documents using temporal language models. We subsequently employ the temporal language models for determining the time of implicit temporal queries, and the determined time is used for re-ranking search results in order to improve the retrieval effectiveness.

We study the effect of terminology changes over time and propose an approach to handling terminology changes using time-based synonyms. In addition, we propose different methods for predicting the effectiveness of temporal queries, so that a particular query enhancement technique can be performed to improve the overall performance. When the time dimension is incorporated into ranking, documents will be ranked according to both textual and temporal similarity. In this case, *time uncertainty* should also be taken into account. Thus, we propose a ranking model that considers the time uncertainty, and improve ranking by combining multiple features using learning-to-rank techniques.

Through extensive evaluation, we show that our proposed time-aware approaches outperform traditional retrieval methods and improve the retrieval effectiveness in searching temporal document collections.

Preface

This PhD thesis is submitted to the Norwegian University of Science and Technology (NTNU) for partial fulfilment of the requirements for the degree of philosophiae doctor.

The PhD project has been performed at the Department of Computer and Information Science, NTNU, Trondheim, with Professor Kjetil Nørvåg and with co-supervisors Professor Jon Atle Gulla and Associate Professor Heri Ramampiaro.

The PhD project is a formal part of LongRec - Records Management over Decades. LongRec was a research project on persistent, reliable and trustworthy long-term archival of digital documents, with emphasis on the availability and use of documents. LongRec was initiated and coordinated by DNV, and performed in cooperation with NTNU, NR, and a number of business partners, with partial funding from the Norwegian Research Council under grant NFR 176818/I40.

Acknowledgements

First of all, I would like to thank my supervisor Professor Kjetil Nørvåg for his excellent supervision and great support during the 4 years of my PhD study. Not only teaching me how to conduct research, he has also given me an insight into my future career. I have enjoyed a lot doing research because of my supervisor and a highly motivating research topic. I would like also thank Professor Jon Atle Gulla and Associate Professor Heri Ramampiaro for being co-supervisors.

In addition, I would give a big thank to the partners of the LongRec project, and DNV in particular, for giving me the opportunity to pursuing my PhD study. Too many names to mention, but many thanks to my fellow PhD students and colleagues for fun discussions during coffee breaks, and their support and friendship. The department's administrations and technical staffs always kindly assisted me, thank you.

I would like to thank Dr. Ricardo A. Baeza-Yates for giving me the chance for collaborating with Yahoo! and thank Yahoo! colleagues for a warm hospitality in Barcelona. A special thank to Dr. Hugo Zaragoza for his mentoring and valuable research discussions during my 3-month internship. Many thanks to Dr. Klaus Berberich, Dr. Roi Blanco and Michael Matthews for kindly collaborating with me. Two mentors from the SIGIR'2009 doctoral consortium, Professor Alistair Moffat and Professor Arjen P. de Vries, also deserve my thanks for useful discussions and encouragements.

I could not have survived living here without a great support from the Bangkok Café team and Thai folks in Trondheim. Finally, I would like to thank my family for their love and moral support, and especially Víctor for inspiration and always being there when I needed.

Contents

Part I

Overview and Background

Chapter 1

Introduction

This PhD thesis addresses different challenges in searching *temporal document collections*, where documents are created and/or edited over time, and the contents of documents are strongly time-dependent. Examples of temporal document collections are web archives, news archives, blogs, personal emails and enterprise documents. The main focus of the PhD thesis is *how to exploit temporal information provided in documents, queries, and external sources of data in order to improve the effectiveness in searching temporal document collections.*

This chapter describes the motivation and research questions addressed in the thesis. In addition, we explain our research context and methods in conducting the PhD work. Our contributions to this thesis are composed of different approaches to solving the addressed research questions. In the end of this chapter, we present the organization of the rest of the thesis.

1.1 Motivation

In this thesis, we address major challenges in searching *temporal document collections*. In such collections, documents are created and/or edited over time. Examples of temporal document collections are web archives, news archives, blogs, personal emails and enterprise documents. Unfortunately, traditional IR approaches based on term-matching only can give unsatisfactory results when searching temporal document collections. The reason for this is twofold: the contents of documents are strongly time-dependent, i.e., documents are about events happened at particular time periods, and a query representing an information need can be time-dependent as well, i.e., a *temporal query*.

One problem faced when searching temporal document collections is the large number of documents possibly accumulated over time, which could result in the large number of irrelevant documents in a set of retrieved documents. Therefore, a user might have to spend more time in exploring retrieved documents in order to find documents satisfying his/her information need. A possible solution for this problem is to take into account the time dimension, i.e. extending keyword search with the creation or published date of

documents. In that way, a search system will narrow down search results by retrieving documents according to both text and temporal criteria, e.g., *temporal text-containment search* [93].

In the rest of this section, we will explain our motivation by presenting some shortcomings of existing document archive search systems, i.e., the Wayback Machine [127] and Google News Archive Search [37].

Wayback Machine

The Wayback Machine [127] is a web archive search tool that is provided by the Internet Archive [49]. The Internet Archive is a non-profit organization with the goal of preserving digital document collections as *cultural heritage* and making them freely accessible online. The Wayback Machine provides the ability to retrieve and access web pages stored in a web archive, and it requires a user to represent his/her information need by specifying the URL of a web page to be retrieved.

For example, given the query URL `http://www.ntnu.no`, the results of retrieval are displayed in a calendar view as depicted in Figure 1.1, which displays the number of times the URL `http://www.ntnu.no` was crawled by the Wayback Machine (not how many times the site was actually updated). Two major problems of using the Wayback Machine are observable. First, it is inconvenient for a user to specify a URL as a query. Second, there is no easy way to sort search results returned by the tool because the results displayed in a timeline according to their crawled dates.

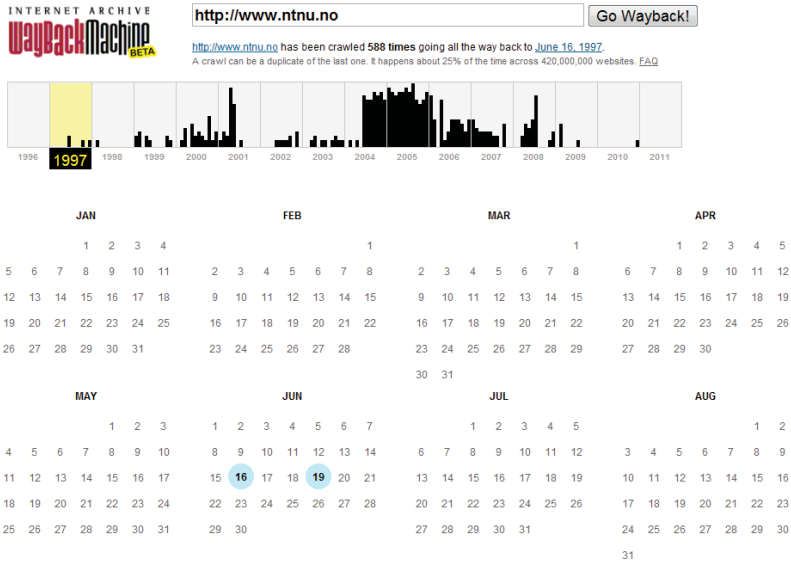


Figure 1.1: Search results of the query URL `http://www.ntnu.no` are displayed in a calendar view (retrieved 2011/08/29).

Google News Archive Search

The Google News Archive Search [37] tool allows a user to search a news archive using a keyword query and a date range. In addition, the tool provides the ability to rank search results by relevance or date. However, there is a problem that has not been addressed by this tool yet, e.g., the effect of terminology changes over time. Consider the following example, a user wants to search for news about Pope Benedict XVI that are written between 2005. So, the user issues the query Pope Benedict XVI and specifies the temporal criteria 2002/01/01 to 2004/31/12. As shown in Figure 1.2, only a small number of documents are returned by the tool where most of them are not relevant to the Pope Benedict XVI. In other words, this problem can be viewed as *vocabulary mismatch*, which is caused by the fact that the term Pope Benedict XVI was not widely used before 2005/04/19 (the date which his papacy began).

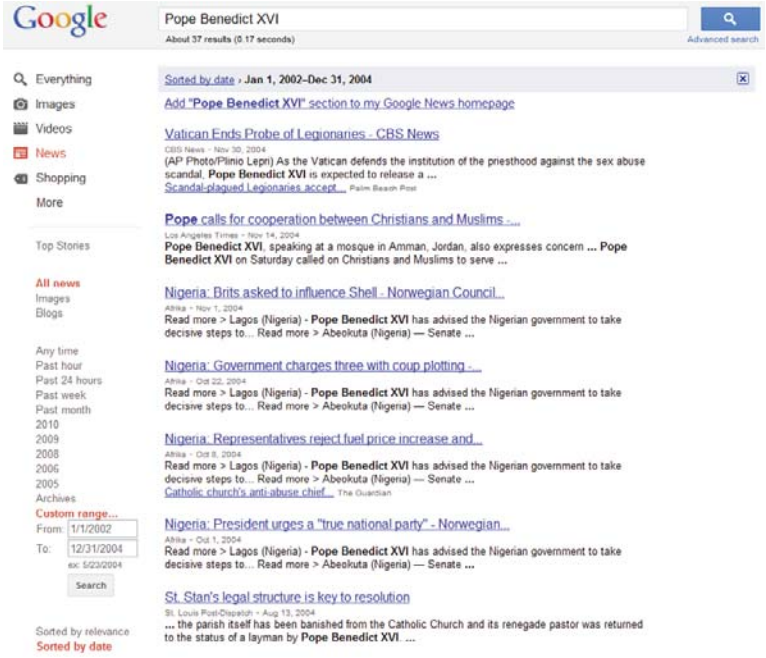


Figure 1.2: Results of the query Pope Benedict XVI and the temporal criteria 2002/01/01 TO 2004/31/12 (retrieved 2011/08/29).

As illustrated by the two examples, it is clear that there is a need for highly efficient and practical approaches to searching temporal document collections. Thus, the goal of this thesis is to identify and study problems in searching temporal document collections, as well as propose approaches as solutions to the problems. In the next section, we will present research problems that are addressed in this thesis.

1.2 Research Questions

Based on the motivation stated in the previous section, the main research question is: *how to exploit temporal information provided in documents, queries, and external sources of data in order to improve the retrieval effectiveness in searching temporal document collections?* Intuitively, we want to solve the main research question by 1) analyzing the contents of temporal document collections, 2) performing an analysis of temporal queries, and 3) explicitly modeling the time dimension into retrieval and ranking. Hence, the research questions we address are corresponding to three topics in information retrieval: *content analysis*, *query analysis*, and *retrieval and ranking models*. More specific research questions are presented below.

1.2.1 Content Analysis

Incorporating the time dimension into search can increase the retrieval effectiveness if information about the creation or publication time of documents is available. However, it is not always easy to find an accurate and trustworthy timestamp of a document for some reasons. First, the time metadata of documents preserved in the past might not be readable and interpretable today. Second, it is difficult to find an accurate and trustworthy timestamp for a web document because of the decentralized nature of the web, where the document can be relocated and its time metadata made unreliable. Moreover, in a web warehouse or a web archive there is no guarantee that a document's creation date and the time of retrieval by the crawler are related. In this thesis, we want to analyze documents' contents in order to estimate the time of publication of documents/contents or the time of the topic of documents' contents. Thus, the first research question we address is:

> *RQ1. How to determine the time of non-timestamped documents in order to improve the effectiveness in searching temporal document collections?*

1.2.2 Query Analysis

Several studies of real-world user query logs have shown that temporal queries comprises a significant fraction of web search queries [87, 95, 141]. For example, Zhang et al. [141] showed that 13.8% of queries contain explicit time and 17.1% of queries have temporal intent implicitly provided. An example of a query with time explicitly provided is U.S. Presidential election 2008, while Germany FIFA World Cup is a query without temporal criteria provided. However, for the latter example, a user's temporal intent is *implicitly* provided, i.e., referring to the World Cup event in 2006. In this thesis, we want to determine the time of a query when time is implicitly provided. Note that, this search scenario happens when users have no knowledge regarding all relevant time periods for a query, so that no time can be explicitly provided in the query. Hence, the second research question we address is:

> *RQ2. How to determine the time of an implicit temporal query and use the determined time for re-ranking search results?*

The effect of terminology changes over time can cause a problem in searching temporal document collections. In fact, the definition, meaning or name of terms can alter. Moreover, terms can be obsolete, i.e., no longer used. For example, the term "Siam" was used as a name for "Thailand" before 1939, but it is rarely used nowadays. This causes a problem for a temporal search if a query and documents are represented in different forms, i.e., historical or modern forms. Given the query Thailand before 1939, documents about Thailand that were written using the term "Siam" and published before 1939 will not be retrieved. Therefore, the third research question we address in this thesis is:

RQ3. How to handle terminology changes in searching temporal document collections?

The research questions presented above are related to query expansion and query reformation. In addition to that, we also want to analyze the retrieval effectiveness of temporal queries with respect to a specific retrieval model. In particular, we will study *query performance prediction* [20] for temporal queries.

Query performance prediction refers to the task of predicting the retrieval effectiveness that queries will achieve with respect to a particular ranking model in advance of, or during the retrieval stage, so that particular actions can be taken to improve the overall performance [43]. Query performance prediction is useful for choosing between alternative query enhancement techniques, e.g., query expansion and query suggestion. In this thesis, we want to investigate different methods for predicting the query performance or retrieval effectiveness of temporal queries. Hence, the fourth research question we address in this thesis is:

RQ4. How to predict the retrieval effectiveness of temporal queries?

Two time dimensions commonly exploited in time-aware ranking are 1) *publication time*, and 2) *content time* (temporal expressions mentioned in documents' contents). As shown later in the thesis, it makes a difference in retrieval effectiveness for temporal queries when ranking using publication time or content time. By determining whether a temporal query is sensitive to publication time or content time, the most suitable retrieval model can be employed. Consider the following examples: given the query Japan quake 869 AD, relevant documents should be those containing the temporal expression 869 AD, but not those created or published in 869 AD. On the other hand, when searching for a current event, such as, academy award rumors, temporal expressions in documents should be more important in consideration than the publication time of documents. Thus, the fifth research question we address is:

RQ5. How to predict the suitable time-aware ranking model for a temporal query?

1.2.3 Retrieval and Ranking Models

In many cases, when searching temporal document collections, search results are displayed in chronological order where recently created documents are ranked higher than older documents. However, chronological ordering is not always effective. Therefore, a retrieval model should rank documents by the degree of relevance with respect to time. More precisely, documents must be ranked according to both textual and temporal similarity. In addition, a time-aware ranking model should also take into account *time uncertainty*, which captures the fact that the relevance of documents may change over time. Thus, the sixth research question we address is:

RQ6. How to explicitly model the time dimension into retrieval and ranking?

In general, a time-aware ranking model gives scores to documents with respect to textual and temporal similarity. However, we want to study whether exploiting other features together with time can help improving the retrieval effectiveness in searching temporal document collections. Specifically, we set up a new task called *ranking related news predictions*, which is aimed at retrieving predictions related to a news story being read, and ranking them according to their relevance to the news story. The challenges of this task are related to various aspects of IR problems: time-aware ranking, sentence retrieval, entity ranking, and domain-specific predictions. In this case, we need to find features used for capturing the similarity between an information need and predictions of future-related events, and combine such features for relevance ranking. Thus, the last research question we address in this thesis is:

RQ7. How to combine different features with time in order to improve relevance ranking?

1.3 Research Context

The PhD work is carried out as a part of four-year PhD program at the Department of Computer and Information Science, Norwegian University of Science and Technology (NTNU) under the main supervision by Professor Kjetil Nørvåg, and the co-supervision by Professor Jon Atle Gulla and Associate Professor Heri Ramampiaro.

The PhD work is a formal part of LongRec - Records Management over Decades [80]. LongRec is a joint-industry project focusing on the challenges of persistent, reliable, and trustworthy long-term storage of digital records. LongRec is organized as a consortium led by DNV and partially funded by the Norwegian Research Council. It emphasizes on the availability and use of information. Problems associated with digital preservation typically emerge when the lifetime of digital documents exceeds 10 years and digital documents are expected to undergo several changes during their lifetime.

1.4 Research Method

We have already explained our motivation and mentioned specific research questions in the previous section. This section presents the research method for doing the PhD work.

We begin our research work by doing a literature study of state-of-the-art of research topics including information retrieval techniques, machine learning, text mining, and information extraction. We aim at analyzing advantages and disadvantages of existing approaches as well as looking for a possibility for improvement.

In order to answer our research questions, we implement an approach for solving a particular research question either as an independent module or as a complete prototype. Then, we evaluate performance of the approach by conducting experiments using test data. Test data used in the experiments can be standard test collections (TREC, CLEF, etc.), or synthetic collections created by us. For the latter, we manually collect queries for evaluation, and obtain relevance judgment using expert judges or crowdsourcing. Several metrics are used for measuring the effectiveness of our proposed approaches, for example, standard IR measures like precision, recall and F-measure.

1.5 Contributions

The work on time-aware approaches to information retrieval is a relatively new field of research. Hence, our contributions are a combination of novel approaches and improvements on existing techniques. In the following, we will give a brief summary of our contributions, and indicate the corresponding research questions as well as the subsequent chapters where detailed contributions can be found. In summary, our contributions to the PhD work *accomplish all research questions*, which are listed below:

I. **Content Analysis**

C1. We propose different techniques for determining the time of non-timestamped documents by improving temporal language models (originally proposed by de Jong et al. [29]). The improved techniques that are proposed include semantic-based preprocessing, and incorporating internal and external knowledge into the language models. In addition, we present a tool for determining the time of a non-timestamped document using the proposed techniques.
[*These contributions are solutions to RQ1, which will be discussed in Chapter 3*.]

II. **Query Analysis**

C2. We perform the first study on how to determine the time of queries without temporal criteria provided, and propose techniques for determining time. In addition, we propose an approach to re-ranking search results by incorporating the determined time of queries.
[*These contributions are solutions to RQ2, which will be discussed in Chapter 4*.]

C3. We model Wikipedia as a temporal resource and use it for discovering time-based synonyms. Moreover, we propose a query expansion technique using the discovered time-based synonyms. Finally, we present a news archive search tool that exploits changing synonyms over time.
[*These contributions are solutions to RQ3, which will be discussed in Chapter 5*.]

C4. We perform the first study and analysis of query performance prediction of temporal queries. In particular, we propose different time-based predictors and techniques for combining multiple predictors in order to improve query performance prediction.
[*These contributions are solutions to RQ4, which will be discussed in Chapter 6*.]

C5. We perform the first study on the impact on retrieval effectiveness of two different ranking models that exploit two time dimensions. We propose an approach to predicting the suitable time-aware ranking model based on machine learning techniques, using three classes of features.
[*These contributions are solutions to RQ5, which will be discussed in Chapter 7*.]

III. **Retrieval and Ranking Models**:

C6. We analyze different time-aware ranking methods concerning two main aspects: 1) whether or not time uncertainty is concerned, and 2) whether the publication time or the content time of a document is used in ranking. By conducting extensive experiments, we evaluate the retrieval effectiveness of different time-aware ranking methods.
[*These contributions are solutions to RQ6, which will be discussed in Chapter 8*.]

C7. The first formalization of the *ranking related news predictions* task is given. Moreover, we propose a learned ranking model incorporating four classes of features including term similarity, entity-based similarity, topic similarity, and temporal similarity.
[*These contributions are solutions to RQ7, which will be discussed in Chapter 9*.]

1.6 Publications

Our contributions to this PhD work have been published in several international conferences. Below is given a list of publications and the corresponding chapters where publications are included.

P1. Nattiya Kanhabua and Kjetil Nørvåg, *Improving Temporal Language Models For Determining Time of Non-Timestamped Documents* [58], Proceedings of the 12th European Conference on Research and Advanced Technology for Digital Libraries 2008 (ECDL'2008), Aarhus, Denmark, September 2008.
[*This publication is included in Chapter 3*.]

P2. Nattiya Kanhabua and Kjetil Nørvåg, *Using Temporal Language Models for Document Dating* (demo) [59], Proceedings of the European Conference on Machine

Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD'2009), Bled, Slovenia, September 2009.
[*This publication is included in Chapter 3*.]

P3. Nattiya Kanhabua and Kjetil Nørvåg, *Exploiting Time-based Synonyms in Searching Document Archives* [61], Proceedings of the ACM/IEEE Conference on Digital Libraries (JCDL'2010), Brisbane, Australia, June 2010.
[*This publication is included in Chapter 5*.]

P4. Nattiya Kanhabua and Kjetil Nørvåg, *Determining Time of Queries for Re-ranking Search Results* [60], Proceedings of the 14th European Conference on Research and Advanced Technology for Digital Libraries 2010 (ECDL'2010), Glasgow, Scotland, UK, September 2010.
[*This publication is included in Chapter 4*.]

P5. Nattiya Kanhabua and Kjetil Nørvåg, *QUEST: Query Expansion using Synonyms over Time* (demo) [62], Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD'2010), Barcelona, Spain, September 2010.
[*This publication is included in Chapter 5*.]

P6. Nattiya Kanhabua and Kjetil Nørvåg, *A Comparison of Time-aware Ranking Methods* (poster) [63], Proceedings of the 34th Annual ACM SIGIR Conference (SIGIR'2011), Beijing, China, July 2011.
[*This publication is included in Chapter 8*.]

P7. Nattiya Kanhabua and Kjetil Nørvåg, *Time-based Query Performance Predictors* (poster) [64], Proceedings of the 34th Annual ACM SIGIR Conference (SIGIR'2011), Beijing, China, July 2011.
[*This publication is included in Chapter 6*.]

P8. Nattiya Kanhabua, Roi Blanco and Michael Matthews, *Ranking Related News Predictions* [57], Proceedings of the 34th Annual ACM SIGIR Conference (SIGIR'2011), Beijing, China, July 2011.
[*This publication is included in Chapter 9*.]

P9. Nattiya Kanhabua, Klaus Berberich and Kjetil Nørvåg, *Time-aware Ranking Prediction*, (under submission).
[*This publication is included in Chapter 7*.]

1.7 Thesis Organization

The thesis is divided into four main parts. Part I presents motivations, research questions, technical background and the state-of-the-art. Part II-VI presents our proposed time-aware approaches to searching temporal document collections. The detailed organization of the thesis is outlined below.

Part I - Overview and Background

Chapter 1 includes this introduction, which states our motivation of this PhD thesis. Research questions and methodology for conducting the thesis are also discussed in this chapter.

Chapter 2 describes technical background composed of fundamental techniques useful for understanding the work in this thesis. In addition, the state-of-the-art relevant to the PhD thesis is also explained.

Part II - Content Analysis

Chapter 3 presents and evaluates our proposed approach to determining the time of non-timestamped documents.

Part III - Query Analysis

Chapter 4 describes approaches to determining the time of queries without temporal criteria provided and evaluate our proposed approaches.

Chapter 5 presents the effect of terminology changes over time, an approach to solving the problem and the evaluation of the proposed approach.

Chapter 6 discusses query performance prediction for temporal queries, and presents time-based predictors as well as the evaluation of the proposed methods.

Chapter 7 presents and evaluates an approach to predicting the suitable time-aware ranking model based on machine learning techniques, using three classes of features.

Part VI - Retrieval and Ranking Models

Chapter 8 describes an empirical comparison of different time-aware ranking methods.

Chapter 9 presents and evaluates a learned ranking model that combines multiple evidences with time for relevance ranking.

Finally, in Chapter 10, we give conclusions, outline future work, and discuss possible research topics beyond what have been addressed in the thesis.

Chapter 2

Background and State-of-the-art

In this chapter, we briefly describe fundamental techniques in the research area of information retrieval, which are useful for understanding our contributions in the following chapters. Then, we describe temporal information retrieval explaining how *time* can be represented and exploited in IR, and giving an overview of the state-of-the-art techniques in temporal information retrieval.

2.1 Information Retrieval

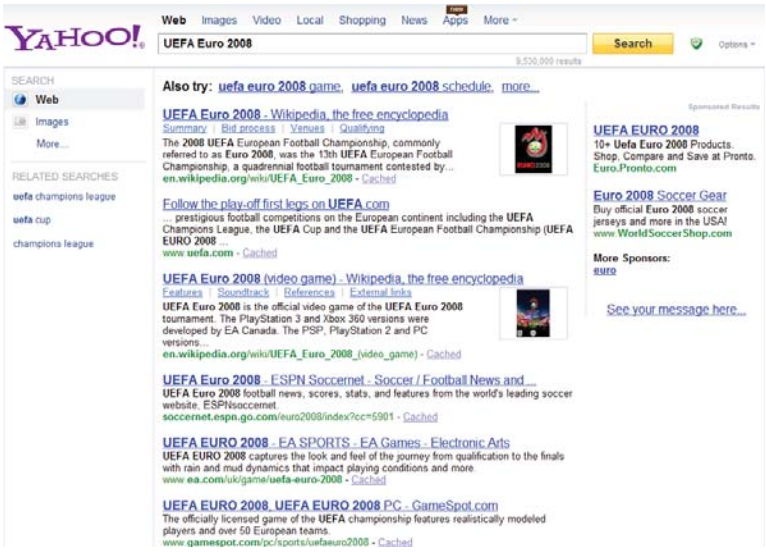
Information retrieval (IR) provides a user with the ability to access information about his/her topics of interest, called *an information need*. A document collection refers to a data repository containing different types of documents, such as textual documents or multimedia documents. A typical IR system allows a user to formulate his/her information need using one or more keywords, called *a query*. Then, the system retrieves documents related to the query and ranks the results according to relevance before returning them to the user. For example, given the query UEFA Euro 2008, the user interfaces and results returned by two different IR systems are shown in Figure 2.1.

In general, the process of information retrieval consists of three main components: document indexing, query processing and document retrieval, as illustrated in Figure 2.2. Another important issue critical to IR is retrieval evaluation, which is not a part of the online retrieval process. We will now describe each IR component in more detail.

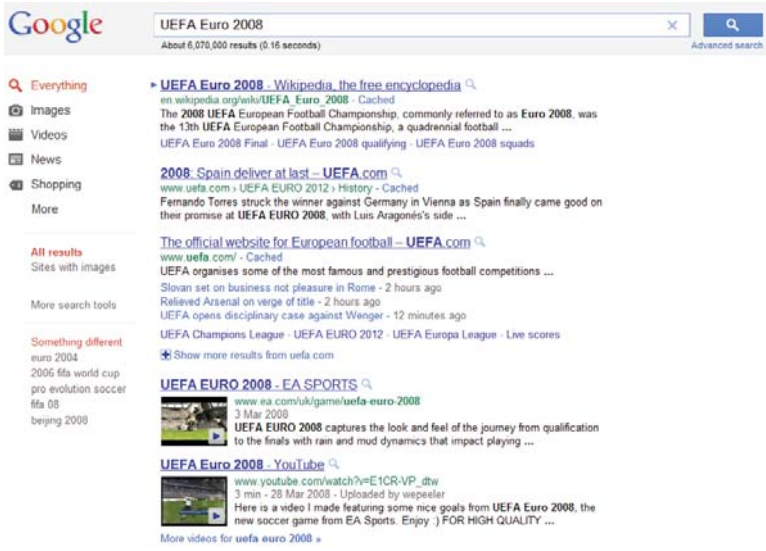
2.1.1 Document Indexing

One major concern when building an IR system is efficiency, that is, the system should process a query and return a result list to the user as fast as possible. In order to increase the speed of search, documents must be indexed. In this way, an IR system avoids linearly scanning the document collection to find the documents matching the query.

The process of transforming documents into index is called *document indexing*, which basically includes two main steps: 1) document acquisition and 2) text preprocessing.



(a) User interface and results from www.yahoo.com



(b) User interface and results from www.google.com

Figure 2.1: Examples of the user interfaces and results returned by two search systems.

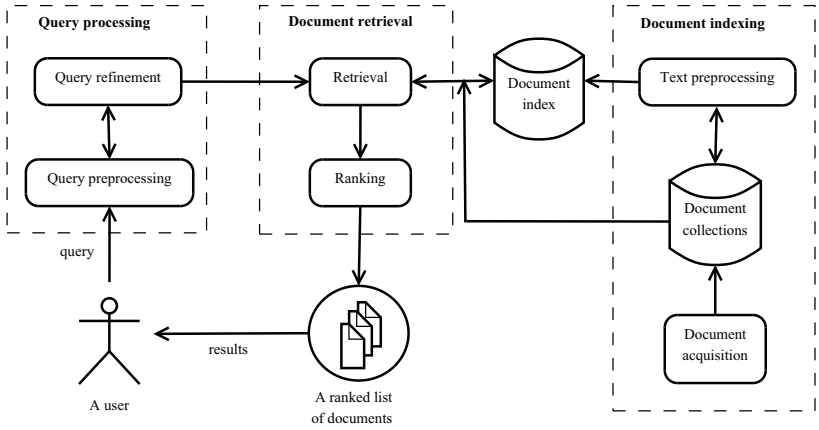


Figure 2.2: Main components of a typical IR system.

Document acquisition refers to the process of obtaining documents, e.g., scanning books into digital documents or crawling web pages. Before a document can be indexed, a text preprocessing step must be performed. For unstructured textual documents, the preprocessing step can include tokenization, part-of-speech tagging, stopword removal, stemming and lemmatization.

1. Tokenization splits a document into a list of words or tokens. In English, a period, a question mark, an exclamation mark, or a comma are used as sentence delimiters.

2. Tagging is the process of labeling each token with its part-of-speech (POS) in the sentence, such as, a noun, a proper noun, an adjective, a verb, a determiner, or a number. POS tagging helps in removal of irrelevant words (e.g., adjectives, or determiners) and also can reduce ambiguity of word senses with several meanings (a noun or a verb). Moreover, tagged tokens are also useful for the stemming process.

3. Stopword removal aims at eliminating less informative or useless words before indexing. Highly frequent words like articles, prepositions, and conjunctions are stopwords which are not necessarily useful in distinguish among documents.

4. The stemming process reduces syntactic variations of words by transforming them into a common form (a root of word, or stem) for example, ‘cars’ becomes ‘car’. In addition, stemming helps in reducing the vocabulary size. An easy and efficient method for stemming is to do affix removal by writing rules.

5. Lemmatization is the lexicon approach mapping inflections of words into one canonical representation or lemma, such as, mapping various verb forms to infinite, mapping plural noun to singular form, or mapping comparative forms of an adjective to the normal form.

After text preprocessing, terms as well as the information about documents and positions will be stored in the *document index*.

2.1.2 Query Processing

A user expresses his/her information need by formulating a query typically consisting of one or more keywords. The results of a query in an IR system can *partly match* a given query. In other words, it retrieves documents containing information *relevant* to the query. A document is considered relevant if it is one that the user perceives as containing information of value with respect to their personal information need [84]. Given a query q and a document d, the degree of relevance of d with respect to q is determined by the IR system and depends on the retrieval model that the system employs. Different retrieval models will be described in the next section. The basic components for query processing are 1) query preprocessing and 2) query refinement. Note that, query refinement is optional, and it is dependent on an IR application. A query must be preprocessed in the same way as the documents in order to be able to match the query with index terms. For instance, a query can be tokenized, stop-word removal, stemmed or lemmatized. In general, a query is not preprocessed extensively because it only consists of a few keywords. After representing both a query and a document using the same format, the matching process will be performed during retrieval.

Consider two example queries: the query car is unable to match a document containing "automobile", and similarly the query plane is unable to match a document containing "aircraft" because documents do not exactly contain the query queries. This is one of two classic problems in natural languages: synonymy and polysemy. Synonymy refers to a case where two different words have the same meaning, e.g., car and "automobile", or plane and "aircraft". On the contrary, polysemy refers to the case where a term has multiple meanings. For instance, the term "java" can refer to programming language, coffee, or an island in Indonesia. In order to overcome the problems, *query refinement* or the process of reformulating the query using semantically similar terms, can be performed either manually by a user or automatically by a system. Two main approaches can be applied to tackle with the problems [84]: 1) global methods and 2) local methods. Global methods reformulate the original query by expanding it with other semantically similar terms, which can be done independently of the initial retrieved results. Examples of global methods are query expansion/reformulation with a thesaurus, spelling correction, and query suggestion. Local methods reformulate the original query by analyzing the initial results returned. The local methods include relevance feedback and pseudo relevance feedback (also known as blind relevance feedback).

In this work, we employ two techniques: *query expansion* using a thesaurus and *pseudo relevance feedback*. Query expansion is aimed at improving the retrieval effectiveness, especially recall, by expanding the query using synonyms or related terms from a thesaurus (or a controlled vocabulary). Generally, a thesaurus is composed of synonymous names for concepts and can be manually created by human editors, semi-automatically created using machine learning, or fully automated using word co-occurrence statistics or query log analysis. Note that, applying query expansion can decrease preci-

Table 2.1: A term-document matrix represents a document using binary weighting $\{1, 0\}$.

		Documents			
		d_1	d_2	d_3	d_4
Terms	UEFA	1	0	0	1
	Europe	1	1	0	1
	football	1	1	1	1
	championship	0	1	1	0

sion significantly when a query contains ambiguous terms.

Relevance feedback is the process of involving a user in improving the final results of retrieval. First, a user issues a query and the system returns the initial results of retrieval. Then, the user is able to provide feedbacks by labeling each document in the initial result set as *relevant* or *non-relevance*. Finally, the system will employ the feedback to reformulate the original query and return the final results, which are retrieved with respect to the modified query. *Pseudo relevance feedback* on the other hand does not require involvement from the user. It assumes that the top-k retrieved documents are relevant to the query without asking for an additional input from the user. Both relevance feedback and pseudo relevance feedback have been shown to improve the retrieval effectiveness. However, they can lead to *query drift* for some queries with too few relevant documents in the top-k retrieved results.

2.1.3 Document Retrieval

Document retrieval is the core process of IR, and a retrieval model is a major component of document retrieval. Several retrieval models have been proposed, for example, Boolean retrieval model, vector space model, probabilistic model, language modeling approaches and learning-to-rank. Retrieval models differ from each other in many aspects including query interpretation, document representation, and document scoring and ranking algorithms employed. In the following, we will explain each of the retrieval models.

Boolean Retrieval Model

The Boolean retrieval model is the simplest IR model. A query is a combination of terms and Boolean operators AND, OR and NOT. A document is modeled as *bag of words* (an unordered list of terms). Each term in the document is represented using binary weighting $\{1, 0\}$ (1 for term presence and 0 for term absence) as illustrated using a term-document matrix in Table 2.1.

The Boolean retrieval model ignores the degree of relevance since it assumes two outcomes of relevance, i.e., relevant or non-relevant. Let $sim(d, q)$ be a function giving a relevance score, a document score is either 1 (relevant) or 0 (non-relevant), that is, $sim(q, d) \in \{1, 0\}$. Given the Boolean query (UEFA AND championship) NOT league, the results are those documents containing both terms "UEFA" and "championship" but not the term "league", as illustrated using a Venn diagram in Figure 2.3. Intuitively, the

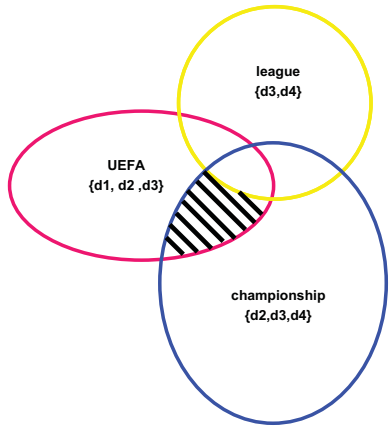


Figure 2.3: Result of the query (UEFA AND championship) NOT league is the shaded area, or d_2.

model returns all documents "exactly matched" with the query terms without ordering the documents.

Despite its simplicity, the retrieval effectiveness of a Boolean query depends entirely on the user. In order to gain high effectiveness, the user can issue a complex query, but it is quite difficult to formulate. If a simple query is used, there might be too few or too many documents retrieved. If a large number of documents are retrieved, this poses a problem for the user because he/she has to spend time looking for those satisfying the information needs.

Vector Space Model

The vector space model is a ranked retrieval model. That is, documents are retrieved and ranked descendingly by the degree of relevance, which can be measured as the similarity between a query and a document. First, a query and documents are represented as vectors of term weights by using a term weighting scheme, e.g., *tf-idf*. Given a term w and a document d, *tf* is the term frequency of w, which is normalized by the total term frequency in d. Thus, *tf* can be computed as:

$$tf(w,d) = \frac{freq(w,d)}{\sum_{j=1}^{n_d} freq(w_j,d)} \tag{2.1}$$

where $freq(w,d)$ is the term frequency of w in d and n_d is the number of distinct terms in d. *tf* captures the importance of a term w in a document by assuming that the higher *tf* score of w, the more importance of w with respect to d. Intuitively, terms that convey the topics of a document should have high values of *tf*.

idf is the inverse document frequency weight of a term w. It measures the importance of w with respect to a document collection. *idf* can be seen as a discriminating property, where a term that appears in many documents is *less discriminative* than a term appears

in a few documents. For example, the term "football" occurring in all documents. Thus, it is *less discriminative* compared to the term "UEFA" occurring in only two documents. *idf* can be computed as:

$$\textit{idf}(w) = \log \frac{N}{n_w} \tag{2.2}$$

where N is the total number of documents in a collection, and n_w is the number of documents in which a term w occurs. Finally, a *tf-idf* weight of a term w in a document d can be computed using the function *tf-idf*(w,d) given as:

$$\textit{tf-idf}(w, d) = \textit{tf}(w, d) \cdot \textit{idf}(w) \tag{2.3}$$

Finally, a query q and a document d can be represented as vectors of *tf-idf* weights of all terms in the vocabulary as:

$$\vec{q} = \langle \psi_{1,q}, \ldots, \psi_{n,q} \rangle$$
$$\vec{d} = \langle \psi_{1,d}, \ldots, \psi_{n,d} \rangle$$

where $\psi_{i,q}$ is *tf-idf* weight of a term w_i in q and $\psi_{i,d}$ is *tf-idf* weight of a term w_i in d. The similarity of the term-weight vectors of q and d can be computed using the cosine similarity as:

$$\begin{aligned} \textit{sim}(\vec{q}, \vec{d}) &= \frac{\vec{q} \bullet \vec{d}}{|\vec{q}| \times |\vec{d}|} \\ &= \frac{\sum_{i=1}^{n} \psi_{i,q} \times \psi_{i,d}}{\sqrt{\sum_{i=1}^{n} \psi_{i,q}^2 \times \sum_{i=1}^{n} \psi_{i,d}^2}} \end{aligned} \tag{2.4}$$

The advantages of the vector space model over the Boolean retrieval model are: 1) it employs term weighting which improves the retrieval effectiveness, 2) the degree of similarity allows partially matching documents to be retrieved, and 3) it is fast and easy for implementing. However, there are some disadvantages of the vector space model. First, it makes no assumption about term dependency, which might lead to poor results [8]. In addition, the vector space model makes no explicit definition of *relevance*. In other words, there is no assumption about whether relevance is binary or mutivalued, which can impact the effectiveness of ranking models.

Probabilistic Model

The probabilistic model was first proposed by Robertson and Jones [103]. The model exploits probabilistic theory to capture the uncertainty in the IR process. That is, documents are ranked according to the probability of relevance. There are two assumptions in this model: 1) relevance is a binary property, that is, a document is either relevant or non-relevant, and 2) the relevance of a document does not depend on other documents.

Given a query q, let R and $\bar{R}$ be the set of relevant documents and the set of non-relevant documents with respect to q respectively. A basic task is to gather all possible evidences in order to describe the properties of the sets of relevant documents and non-relevant documents. The similarity of q and a document d can be computed using the odd ratio of relevance as:

$$sim(d, q) = \frac{P(R|d)}{P(\bar{R}|d)} \tag{2.5}$$

In order to simplify the calculation, Bayes' theorem is applied yielding the following formula:

$$\begin{aligned} sim(d, q) &= \frac{P(R|d)}{P(\bar{R}|d)} \\ &= \frac{P(R) \cdot P(d|R)}{P(\bar{R}) \cdot P(d|\bar{R})} \\ &\approx \frac{P(d|R)}{P(d|\bar{R})} \end{aligned} \tag{2.6}$$

where $P(R)$ is the a prior probability of a relevant document, and $P(\bar{R})$ is the a prior probability of a non-relevant document. For a given query q, it is assumed that both prior probabilities are the same for all documents, so they can be ignored from the calculation. $P(d|R)$ and $P(d|\bar{R})$ are probabilities of randomly selecting a document d from the set of relevant documents R and the set of non-relevant documents $\bar{R}$ respectively.

In the probabilistic model, a document d is represented as a vector of terms with binary weighting, which indicates term occurrence or non-occurrence.

$$\vec{d} = \langle \psi_{1,d}, \ldots, \psi_{n,d} \rangle$$

where $\psi_{i,d}$ is the weight of a term w_i in a document d, and $\psi_{i,d} \in \{0, 1\}$. In order to compute $P(d|R)$ and $P(d|\bar{R})$, it assumes the Naive Bayes conditional independence [84], that is, the presence or absence of a term in a document is independent of the presence or absence of other terms in the given query. Thus, the computation of similarity can be simplified as:

$$\begin{aligned} sim(d, q) &\approx \frac{P(d|R)}{P(d|\bar{R})} \\ &\approx \frac{\prod_{i=1}^{n} P(w_i|R)}{\prod_{i=1}^{n} P(w_i|\bar{R})} \end{aligned} \tag{2.7}$$

where $P(w_i|R)$ is the probability that a term w_i occurs in relevant documents, and $P(w_i|\bar{R})$ is the probability that a term w_i occurs in non-relevant documents. By modeling relevance using probability theory makes the probabilistic model theoretically sound compared to the Boolean retrieval model and the vector space model. However, a drawback is an independence assumption of terms, which is contrary to the fact that any two terms can

Table 2.2: Example of a language model or a probability distribution over terms in the language.

Term	Probability
UEFA	0.18
Europe	0.27
championship	0.36
football	0.18

be semantically related. In addition, the probabilistic model is difficult to implement because the complete sets of relevant documents and non-relevant documents are not easy to obtain. Thus, in order to compute $P(w_i|R)$ and $P(w_i|\bar{R})$, it is needed to guess prior probabilities of a term w_i by retrieving top-n relevant documents and then perform iterative retrieval in order to recalculate probabilities. This makes it difficult to implement the model. In addition, the probabilistic model ignores the frequency of terms in a document.

Language Modeling

Originally, language modeling was employed in speech recognition for recognizing or generating a sequence of terms. In recent years, language model approaches have gained interests from the IR community and been applied for IR. A language model M_D is estimated from a set of documents D, which is viewed as the probability distribution for generating a sequence of terms in a language. An example of a language model is shown in Table 2.2. The probability of generating a sequence of terms can be computed by multiplying the probability of generating each term in the sequence (called a unigram language model), which can be computed as:

$$P(w_1, w_2, w_3|M_D) = P(w_1|M_D) \cdot P(w_2|M_D) \cdot P(w_3|M_D) \tag{2.8}$$

The original language modeling approach to IR is called the *query likelihood model* [84]. In this model, a document d is ranked by the probability of a document d as the likelihood that it is relevant to a query q, or $P(d|q)$. By applying Bayes' theorem, $P(d|q)$ can be computed as:

$$P(d|q) = \frac{P(q|d) \cdot P(d)}{P(q)} \tag{2.9}$$

where $P(q)$ is the probability of a query q, and $P(d)$ is a document's prior probability. Both $P(q)$ and $P(d)$ are in general ignored from the calculation because they have the same values for all documents. The core of the *query likelihood model* is to compute $P(q|d)$ or the probability of generating q given the language model of d, M_D. $P(q|d)$ can be computed using maximum likelihood estimation (MLE) and the unigram assumption

as:

$$\begin{aligned} P(q|M_d) &= P(w_1, w_2, w3|M_d) \\ &= \prod_{i=1}^{n_q} P(w_i|M_d) \end{aligned} \tag{2.10}$$

where n_q is the number of terms in q. The equation above is prone to zero-probability, which means that one or more terms in q may be absent from a document d. In order to avoid zero-probability, a smoothing technique can be applied in order to add a small (non-zero) probability to terms that are absent from a document. Such a small probability is generally taken from the background document collection. For each query term w, a smoothing technique is applied yielding the estimated probability $\hat{P}(w|d)$ of generating each query term w from d as:

$$\hat{P}(w|d) = \lambda \cdot P(w|M_d) + (1 - \lambda) \cdot P(w|M_C) \tag{2.11}$$

where the smoothing parameter $\lambda \in [0, 1]$. C is the background document collection. M_C is the language model generated from the background collection.

Learning to Rank

Many researchers have applied machine learning algorithms in order to optimize the quality of ranking, called learning-to-rank approaches. In general, there are three main steps for modeling a ranking function using learning-to-rank approaches [77]:

1. *Identify features*. A set of features $\{x_1, x_2, \ldots, x_m\}$ are defined as sources of the relevance of a document d_i with respect to a query q_j. Normally, a value of each feature x_i is a real number between $[0, 1]$. The same notation will be used for both feature and its value, that is x_i. Given a query q_j, a document d_i can be represented as a vector of feature values, $d_i = (x_1, x_2, \ldots, x_z)$ indicating the relevance of d_i with respect to q_j.

2. *Learn a ranking model*. Machine learning is used for learning a ranking function $h(q, d)$ based on training data, called supervised learning. Training data is a set of triples of labeled or judged query/document pairs $\{(q_j, d_i, y_k)\}$, where each document d_i is represented by its feature values, $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$. A judgment or label y_k can be either relevant or non-relevant $y_k \in \{1, -1\}$, or a *rank* representing by natural numbers $y_k \in \mathbb{N}$.

3. *Rank documents using models*. The ranking function $h(q, d)$ learned in the previous step will be used for ranking test data, or a set of unseen query/document pairs $\{(q_j, d_i)\}$ where $i \notin \{1, \ldots, n\}$ and $j \notin \{1, \ldots, m\}$. The result is a judgment or label y'_k for each query/document pair.

A ranking model $h(d, q)$ is obtained by training a set of labeled query/document pairs using a learning algorithm. A learned ranking model is essentially a weighted coefficient w_i of a feature x_i. An unseen document/query pair (d', q') will be ranked according to a weighted sum of feature scores:

$$score(d', q') = \sum_{i=1}^{N} w_i \times x_i^{q'} \tag{2.12}$$

where N is the number of features. Many existing learning algorithms have been proposed, and can be categorized into three approaches: pointwise, pairwise, and listwise approaches [77]. The pointwise approach assumes that retrieved documents are independent, so it predicts a relevance judgment for each document and ignores the positions of documents in a ranked list. The pairwise approach considers a pair of documents, and relevance prediction is given as the relative order between them (i.e., pairwise preference). The listwise approach considers a whole set of retrieved documents, and predicts the relevance degrees among documents. For a more detailed description of each approach, please refer to [77].

2.1.4 Retrieval Evaluation

In the IR research community, it is common to evaluate an IR system using a test collection, which is composed of various document collections, a set of queries, and relevance judgments for queries. For example, the Text Retrieval Conference (TREC) and Cross Language Evaluation Forum (CLEF) provide test collections and evaluation data for different IR tasks, such as, ad hoc search, enterprise search, question answering, cross language retrieval, etc. Building each test collection involves evaluating all documents in a collection, which is a time-consuming process for human assessors and not feasible in practice. In general, a *pooling* technique is used in TREC for creating a *pool* of relevance judgments [24]. Top-k documents (between 50 and 200) from the rankings obtained by different search engines (or retrieval algorithms) are merged into a pool, and duplicates are removed. Then, documents are presented in some random order to assessors for making relevance judgments. Hence, the final output of pooling is a set of relevance judgments for the queries.

Basically, there are two aspects of evaluating an IR system: 1) efficiency and 2) effectiveness. Efficiency measures a system's response time and space usage, while effectiveness measures the quality of the system's relevance ranking. In this work, we only consider the retrieval effectiveness aspect. Two metrics that are commonly used for evaluating the retrieval effectiveness are *precision* and *recall*. Precision is the fraction of *retrieved* documents that are relevant. Recall is the fraction of *relevant* documents that are retrieved. Let R be the set of relevant documents and A be the set of retrieved documents

(answer set) of q. Precision and recall can be computed as:

$$\begin{aligned} \textit{precision} &= \frac{|R \cap A|}{A} \\ \textit{recall} &= \frac{|R \cap A|}{R} \end{aligned} \tag{2.13}$$

F-measure is a single measure that combines precision and recall, and it is computed as the weighted harmonic mean of precision and recall:

$$F = \frac{2 \cdot P \cdot R}{(P + R)} \tag{2.14}$$

where P is precision and R is recall. In this work, we also use other metrics for measuring the retrieval effectiveness. *Precision at top-k* documents, so-called P@k, focuses on only top documents and it is easy to compute. For example, precision at top-5, 10 and 15 are denoted as P@5, P@10 and P@15 respectively. *Mean Average Precision* (MAP) provides a summarization of rankings from multiple queries by averaging the precision values from the rank positions where a relevant document was retrieved, or *average precision*. *Mean Reciprocal Rank* (MRR) is the average of the reciprocal ranks over a set of queries, where reciprocal rank is the rank at which the first relevant document is retrieved.

More thorough description on retrieval evaluation can be found in [8, 24, 84].

2.2 Temporal Information Retrieval

Temporal information retrieval refers to IR tasks that analyze and exploit the time dimension embedded in documents to provide alternative search features and user experience. Examples of interesting applications of temporal IR are document exploration, similarity search, summarization, and clustering.

As mentioned previously, we want to exploit temporal information in document collections, queries or external sources of data in order to improve the quality of search or the retrieval effectiveness. Basically, two types of temporal information particularly useful for temporal IR: 1) the publication or creation time of a document, and 2) temporal expressions mentioned in a document or a query. In the following, we first give an overview of different types of temporal expressions. Then, we present time models defined in two previous work [10, 29], which later are used as time models also in our work. Finally, we present state-of-the-art in temporal IR.

2.2.1 Temporal Expressions

As explained in [2], there are three types of temporal expressions: explicit, implicit and relative. An explicit temporal expression mentioned in a document can be mapped directly to a time point or interval, such as, dates or years on the Gregorian calendar. For example, “July 04, 2010” or “January 01, 2011” are explicit temporal expressions. An implicit

temporal expression is given in a document as an imprecise time point or interval. For example, “Independence Day 2010” or “New Year Day’s 2011” are implicit expressions that can be mapped to “July 04, 2010” or “January 01, 2011” respectively. A relative temporal expression occurring in a document can be resolved to a time point or interval using a time reference - either an explicit or implicit temporal expressions mentioned in a document or the publication date of the document itself. For example, the expressions “this Monday” or “next month” are relative expressions which we map to exact dates using the publication date of the document.

2.2.2 Models for Time, Documents and Queries

In temporal IR, the time dimension must be explicitly modeled in documents and queries. In the following, we outline models for time, documents and queries that are employed in temporal IR tasks.

Time Models

In [29], de Jong et al. modeled time as a *time partition*, that is, a document collection is partitioned into smaller time periods with respect to a time granularity of interests, e.g. *day*, *week*, *month*, or *year*. A document collection C contains a number of corpus documents, such as, $C = \{d_1, \ldots, d_n\}$. A document d_i is composed of bag-of-words, and the publication time of d_i is represented as $\mathit{Time}(d_i)$. Thus, d_i can be represented as $d_i = \{\{w_1, \ldots, w_n\}, \mathit{Time}(d_i)\}$. Given a time granularity of interest and C is partitioned into smaller time periods, the associated time partition of d_i is a time period $[t_k, t_{k+1}]$ that contains the publication time of d_i, that is $\mathit{Time}(d_i) \in [t_k, t_{k+1}]$. For example, if the time granularity of *year* is used, the associated time interval for 2011/08/22 will be $[2011/01/01, 2010/12/31]$. Two data structures for storing terms and associated time are proposed [29] as shown in Table 2.3. Both data structures have different advantages and disadvantages. A table is good when data is sparse, and it is efficient in sorting. On the other hand, a matrix gives a direct access to data which can improve access time.

In [10], Berberich et al. represented a temporal expression extracted from a document or the publication time of a document as a quadruple: (tb_l, tb_u, te_l, te_u) where tb_l and tb_u are the lower bound and upper bound for the begin boundary of the time interval respectively, which underline the time interval’s earliest and latest possible begin time. Similarly, te_l and te_u are the lower bound and upper bound for the end boundary of the time interval respectively, which underline the time intervalŠs earliest and latest possible end time. Since the time interval is not necessarily known exactly, the time model of Berberich et al. is proposed to capture lower and upper bounds for the interval boundaries. To interpret the time uncertainty in this model, consider the following example given in [10]. The temporal expression “in 1998” is represented as (1998/01/01, 1998/12/31, 1998/01/01, 1998/12/31), which can refer to any time interval $[b, e]$ having a begin point $b \in [tb_l, tb_u]$ and an end point $e \in [te_l, te_u]$ where $b \leq e$. Note that, the actual value of any time point, e.g., tb_l, tb_u, te_l, or te_u, is an integer or the number of time units (e.g., milliseconds or days) passed (or to pass) a reference point of time (e.g., the UNIX epoch).

Table 2.3: Data Models: Table vs. Matrix

(a) Table

Word	Partition	Frequency
terrorist	2002	9478
terrorist	2003	7750
terrorist	2004	5212
tsunami	2002	101
tsunami	2003	56
tsunami	2004	26905
world cup	2002	19273
world cup	2003	6069
world cup	2004	448

(b) Matrix

	2002	2003	2004
terrorist	9478	7750	5212
tsunami	101	56	26905
world cup	19273	6069	448

These time units are referred as *chronons* and a temporal expression t is denoted as the set of time intervals that t can refer to.

Document Model

A document d consists of a textual part d_{text} (an unordered list of terms) and a temporal part d_{time} composed of the publication date and a set of temporal expression $\{t_1, \ldots t_k\}$. The publication date of d can be obtained from the function *PubTime*(d). Temporal expressions mentioned in the contents of d can be obtained from the function *ContentTime*(d). Both the publication date and temporal expressions can be represented using the time models defined above.

Temporal Query Model

A temporal query q refers to a query representing *temporal information needs*, which is composed of two parts: keywords q_{text} and a temporal expression q_{time}. In other words, a user wants to know about documents that are relevant to both the topic of interest and temporal intent. Temporal queries can be categorized into two types: 1) those with time explicitly specified, and 2) those with implicit temporal intents. An example of a query with time explicitly specified is the eruptions of volcanoes in Iceland *before 2010*. In this case, a temporal intent is represented by the temporal expression "before 2010" indicating that a user wants to know about volcanic events in Iceland during the years before 2010. A query of the latter type is, for instance, Europe flight disruptions from ash cloud that contains an implicit temporal intent referring to the Europe air travel disruption caused by the Iceland volcano ash problem in "April 2010" or "May 2011". Similarly, the temporal part of a query or q_{time} can be represented using any time models defined above.

Note that, there is no standard terminology for referring a query in this research area. Previous work [71, 87, 93, 126] mainly uses the term *temporal queries*, however, the term

time-sensitive queries has been used recently in some work [27, 32, 140]. We will use these two terms interchangeably throughout this thesis.

2.2.3 State-of-the-art in Temporal Information Retrieval

In this section, we will give a brief overview of related work in temporal IR: determining time for non-timestamped documents, time-aware ranking, temporal indexing, visualization using a timeline, and searching with the awareness of terminology changes.

Determining Time for Non-timestamped Documents

Determining the time of a document can be done using two methods: learning-based and non-learning methods. The difference between these two methods is that the former determines time of a document by learning from a set of training documents, while the latter does not require a corpus for training. Learning-based methods are presented in [29, 116, 117]. In [116, 117], they use a statistical method (hypothesis testing) on a group of terms having an overlapped time period in order to determine if they are statistically related. If the computed values from testing are above a threshold, those features are coalesced into a single topic, and the time of the topic is estimated from a common time period associated to each term. Other previous work on this topic is the work by de Jong, Rode, and Hiemstra [29] based on a temporal language model, which will be explained in more detail in Chapter 3.

Non-learning methods are presented in [78, 82]. In order to determine time of a document, temporal expressions in the document are annotated and resolved into concrete dates. A relevancy of each date is computed using the frequency of which the date appears in the document. The most relevant date is used as a reference date for the document, however, if all dates are similar relevant, the publication date will be used instead. In the end, the event-time period of the document is generated by assembling all nearly dates to the reference date where their relevancy must be greater than a threshold.

Comparing the non-learning to learning-based methods, both of them return two different aspects of time. The first method gives a summary of time of events appeared in the document content, or time of topic of contents. The second method gives the most likely originated time of the document, or time of document creation.

Time-aware Ranking

Time-aware ranking techniques can be classified into two categories: techniques based on 1) *link-based analysis* and 2) *content-based analysis*. Approaches of the first category exploit the link structures of documents in a ranking process, whereas the latter approach leverages the contents of documents instead of links. In our research context, we will focus on analyzing contents only, because information about links is *not* available in all application domains, and content-based analysis seems to be more practical for search in general.

Previous work on time-aware ranking that exploits link structures is presented in [14, 26, 135]. In [135], Yu et al. pointed out that traditional link-based algorithms (i.e., PageRank and HITS) simply ignore the temporal dimension in ranking. Thus, they modified the PageRank algorithm by taking into account the date of a citation in order to improve the quality of publication search. A publication obtains a ranking score by accumulating the weights of its citations, where each citation receives a weight exponentially decreased by its age. In [14], Berberich et al. also extended PageRank to rank documents with respect to freshness. The difference is that this work defines freshness as a linear function that will give a maximum score when the date of document or link occur within *the user specified period* and decrease a score linearly if it occurs outside the interval. In more recent work [26], Dai and Davison studied the dynamics of web documents and links that can affect relevance ranking, and proposed a link-based ranking method incorporating the freshness of web documents. Intuitively, features used for ranking were captured by considering two temporal aspects: 1) how fresh the page content is, referred to as page freshness, and 2) how much other pages care about the target page, referred as in-link freshness.

Ranking methods based on an analysis of document content are presented in [31, 51, 74, 100, 111]. In [74], Li and Croft proposed to incorporate time into a language modeling framework [73, 101], called a time-based language model. In the previous language model [73, 101], it is assumed uniform prior probabilities, but in the new model, they assigned prior probabilities with an exponential function of the created date of a document where a document with a more recent creation date obtains high probability. In this work, they did not explicitly use the contents of documents, but only date metadata. Jatowt et al. presented in [51] an approach to rank a document by its freshness and relevance. The method analyzed changed contents between a current version with archived versions, and find a similarity score of changes to a query topic. It is assumed that a document is likely to have fresh contents if it is frequently changed and on-topic. Thus, documents are ranked with respect to the relevance of changed contents to the topic, the size of changes and the time difference between consecutive changes. In other words, a document is ranked high if it is modified significantly and recently. In [31], Diaz and Jones used timestamp from document metadata to measure the distribution of retrieved documents and create the temporal profile of a query. They showed that the temporal profile together with contents of retrieved documents improve average precision for the query by using a set of different features for discriminating between temporal profiles: KL divergence, autocorrelation, the kurtosis order, and three factors from the burst model.

Another work on content-based analysis is presented in [100]. Perkiö et al. introduced a process of automatically detecting a topical trend (the strength of a topic over time) within a document corpus by analyzing temporal behavior of documents using a statistic topic model. Then, it is possible to use topical trends on top of any traditional ranking like tf-idf to improve the effectiveness of retrieval. In [111], Shaparenko et al. proposed a method that does not require link information. The proposed method is appropriate for various types of documents, for example, emails or blogs, lacking meaningful citation data. The idea is to identify the most influent document by defining the impact of a document as the amount of follow-up work it generates represented as *lead/lag index*. The

index measures if a document is more leader or more follower by comparing similarities of two documents and time lag.

Temporal Indexing

Also related is work on temporal indexing for supporting temporal text-containment queries. Nørvåg presented in [93] an approach to manage documents and index structures in temporal document databases. Using a web warehouse containing historical web pages as a testing environment, the author showed that different indexing methods proposed improve the performance of temporal text-containment queries. In [12], Berberich et al. presented a method for text search over temporally versioned documents. They proposed the *temporal coalescing* technique for reducing the index size, and proposed the sublist materialization technique to improve index performance concerning space and time. Documents are retrieved according to a query and user's specified time, and are ranked based on tf-idf.

Visualization using a Timeline

Recent work also consider visualization of search results using temporal information to place retrieved documents on a timeline, which is useful for document browsing [1, 2, 37]. When a user enters only keywords as a query, retrieved results are too broad without giving temporal context. To narrow down a set of documents retrieved, it is necessary to give an overview of possible time periods relevant to the query and suggest that as a hint to the user. In [11], they display a histogram of a distribution of the size of estimated results over a timeline. The intention is to draw tentative time periods for the query, and then the user can refine the query with the new temporal context he/she is interested in.

Searching with the Awareness of Terminology Changes

Search results can be affected by the terminology changes over time, for instance, changes of words related to their definitions, semantics, and names (people, location, etc.). It is important to note that language changes is an continuous process that can be observable also in a short term period. The variation in languages causes two problems in text retrieval; 1) *spelling variation* or a difference in spelling between the modern and historic language, and 2) *semantics variation* or terminology evolution over time (new words are introduced, others disappears, or the meaning of words changes).

Previous work [35, 36, 69] addressed the spelling variation problem using techniques from cross language information retrieval (CLIR). In [69], Koolen et al. proposed a cross-language approach to historic document retrieval. A rule-based method for modernizing historic languages, and the retrieval of historic documents using cross-language information retrieval techniques are proposed. In [35, 36], Ernst-Gerlach and Fuhr used probabilistic rule-based approaches to handling term variants when searching historic texts. In this case, a user can search using queries in contemporary language and the issued queries are translated into an old spelling possibly unknown to the user, which is similar to query expansion. As explained in [35], there are two ways to perform query expansion: an

expansion of query and an expansion of index. In the first case, a set of rules is automatically constructed for mapping historic terms into modern terms. In the latter case, based on a lexical database, terms are indexed together with their synonyms and holonyms as additional indices.

The affect of terminology evolution over time is addressed in [13, 28, 55, 56, 118]. In [13], Berberich et al. proposed a method based on a hidden Markov model for reformulating a query into terminology prevalent in the past. Kaluarachchi et al. [55, 56] studied the problem of concepts (or entities) whose names can change over time. They proposed to discover concepts that evolve over time using association rule mining, and used the discovered concepts to translate time-sensitive queries and answered appropriately. In [28], de Boer et al. presented a method for automatically extracting event time periods related to concepts from web documents. In their approach, event time periods are extracted from different documents using regular expressions, such as, numerical notations for years. Tahmasebi et al. [118] proposed to automatically detecting terminology evolution within large, historic document collections by using clustering techniques and analyzing co-occurrence graph.

Part II

Content Analysis

Chapter 3

Determining Time of Non-timestamped Documents

In order to incorporate the time dimension into search, a document should be assigned to its time of creation or published date. However, it is difficult to find an accurate and trustworthy timestamp for a document. This chapter addresses the research problem: *how to determine the time of non-timestamped documents in order to improve the effectiveness in searching temporal document collections?*

3.1 Motivation

When searching temporal document collections, it is difficult to achieve high effectiveness using only a keyword query because the contents of documents are strongly time-dependent. Possible solutions to increase the retrieval effectiveness are, for instance, extending keyword search with the publication time of documents (called temporal criteria), or automatically re-ranking retrieved documents using time. Incorporating the time dimension into search will increase the retrieval effectiveness if a document is assigned to its time of creation or published date. However, due to its decentralized nature and the lack of standards for date and time, it is difficult to find an accurate and trustworthy timestamp for a web document. In a web warehouse or a web archive, there is no guarantee that the creation time and the time of retrieval by a web crawler are related. Similarly, a document can be relocated and its metadata made unreliable. The purpose of determining time for non-timestamped documents is to estimate the time of publication of document/contents or the time of topic of documents' contents. The process of determining the time of documents is called *document dating*.

Contributions

Our main contributions in this chapter are:

- We propose different techniques for improving temporal language models (originally proposed by de Jong et al. [29]) used for determining the creation time of

non-timestamped documents. The proposed approaches include different semantic-based preprocessing. In addition, we aim at improving the quality of document dating by incorporating internal and external knowledge into the temporal language models.

- We present a system prototype for dating documents using the proposed extension approaches. The system prototype can take different formats of input: a file, the contents of a given URL, or directly entered text. As output, it will present an estimation of possible time periods associated with the document, with confidence of each of the estimated time periods.

Organization

The organization of the rest of this chapter is as follows. In Section 3.2, we give an overview of related work. In Section 3.3, we outline preliminaries that will be used as the basis of our approach. In Section 3.4, we explain semantic-based techniques used in data preprocessing. In Section 3.5, we propose three new approaches that improve the previous work: word interpolation, temporal entropy and using external search statistics. In Section 3.6, we evaluate our proposed techniques. In Section 3.7 we describe our document dating prototype, and we demonstrate the usage of the document dating system. Finally, in Section 3.8, we give conclusions.

3.2 Related Work

Previous work on determining the time of a document can be categorized into 2 approaches: learning-based and non-learning methods. The difference between the two methods is that the former determines the time of a document by learning from a set of training documents, while the latter does not require a corpus collection. Learning-based methods are presented in [29, 116, 117]. In [116, 117], they use a statistical method called *hypothesis testing* on a group of terms having an overlapped time period in order to determine if they are statistically related. If the computed values from testing are above a threshold, those features are coalesced into a single topic, and the time of the topic is estimated from a common time period associated to each term. Another method presented by de Jong et al. in [29] is based on a temporal language model where the time of the document is assigned with a certain probability. We will discuss in details the temporal language model in the next section.

Non-learning methods are presented in [78, 82, 94]. They require an explicit time-tagged document. In order to determine the time of a document, each time-tagged word is resolved into a concrete date and a relevancy of the date is computed using the frequency of which the date appears in the document. The most relevant date is used as a reference date for the document, however, if all dates are similar relevant, the publication date will be used instead. In the end, the event-time period of the document is generated by assembling all nearly dates to the reference date where their relevancy must

be greater than a threshold. Nunes et al. [94] propose an alternative approach to dating a non-timestamped document using its neighbors, such as 1) documents containing links to the non-timestamped document (incoming links), 2) documents pointed to the non-timestamped document (outgoing links) and 3) the media assets (e.g., images) associated with the non-timestamped document. They compute the average of last-modified dates extracted from neighbor documents and use it as the time for the non-timestamped document.

More recent work on document dating is the work by Chen et al. [21]. They propose a hybrid approach, i.e., extracting and inferring the timestamp of a web document using a machine learning technique. Different features are used including linguistic features, position-based features and the page format and tag information of web documents. In addition, the links and contents of a web document and its neighbors are also exploited.

Comparing the non-learning to learning-based methods, both of them return two different aspects of time. The first method gives a summary of the time of events appeared in the document content, while the latter one gives the most likely originated time which is similar to written the time of the document. In this chapter, we focus on analyzing contents only because information about links is not available in all domains, and content-based analysis seems to be more practical for a general search application.

3.3 Preliminaries

In this section, we briefly outline our document model and the statistic language model presented by de Jong, Rode and Hiemstra [29]. For short we will in the following denote their approach as the *JRH* approach.

3.3.1 Document Model

In our context, a document collection contains a number of corpus documents defined as $C = \{d_1, d_2, d_3, \ldots, d_n\}$. A document has two views: a logical view and a temporal view. The logical view of each document can be seen as bag-of-words (an unordered list of terms, or features), while the temporal view represents trustworthy timestamps. A simple method of modeling the temporal view is partitioning time spans into a smaller time granularity. A document model is defined as $d_i = \{\{w_1, w_2, w_3, \ldots, w_n\}, (t_i, t_{i+1})\}$ where $t_i < t_{i+1}$, $t_i < \textit{PubTime}(d_i) < t_{i+1}$, and (t_i, t_{i+1}) is the temporal view of the document which can be represented by a time partition. $\textit{PubTime}(d_i)$ is a function that gives trustworthy timestamp of the document and must be valid within in the time partition.

3.3.2 Temporal Language Models

The *JRH* approach is based on temporal language models, which incorporates the time dimension into language modeling [101]. The temporal language models assign a probability to a document according to word usage statistics over time. The *JRH* approach

employs a normalized log-likelihood ratio (NLLR) [70] for computing the similarity between two language models. Given a partitioned corpus, it is possible to determine the time of a non-timestamped document d_i by comparing the language model of d_i with each corpus partition p_j using the following equation:

$$\mathit{NLLR}(d_i, p_j) = \sum_{w \in d_i} P(w|d_i) \times \log \frac{P(w|p_j)}{P(w|C)} \tag{3.1}$$

$$P(w|d_i) = \frac{\mathit{tf}(w, d_i)}{\sum_{w' \in d_i} \mathit{tf}(w', d_i)} \tag{3.2}$$

$$P(w|p_j) = \frac{\mathit{tf}(w, p_j)}{\sum_{w' \in p_j} \mathit{tf}(w', p_j)} \tag{3.3}$$

$$P(w|C) = \frac{\mathit{tf}(w, C)}{\sum_{w' \in C} \mathit{tf}(w', C)} \tag{3.4}$$

where $\mathit{tf}(w, d_i)$ is the frequency of a term w in a non-timestamped document d_i. $\mathit{tf}(w, p_j)$ is the frequency of w in a time partition p_j. $\mathit{tf}(w, C)$ is the frequency of w in the entire collection C. In other word, C is the background model estimated on the entire collection. The timestamp of the document is the time partition which maximizes the score according to the equation above. The intuition behind the described method is that given a document with unknown timestamp, it is possible to find the time interval that mostly overlaps in term usage with the document. For example, if the document contains the word "tsunami" and corpus statistic shows this word was very frequently used in 2004/2005, it can be assumed that this time period is a good candidate for the document timestamp.

As can be seen from the equation, words with zero probability are problematic, and smoothing (linear interpolation [70] and Dirichlet smoothing [139]) is used to solve the problem by giving a small (non-zero) probability to words absent from a time partition. In the next section, we will present out approach to determining the time of a document, which basically extends the *JRH* approach.

3.4 Semantic-based Preprocessing

Determining the time of a document from a direct comparison between extracted words and corpus partitions has limited accuracy. In order to improve the performance, we propose to integrate semantic-based techniques into document preprocessing. We have in our work used the following techniques:

- **Part-of-Speech Tagging:** Part-of-speech (POS) tagging is the process of labeling a word with a syntactic class. In our work, we use POS tagging to select only the most interesting classes of words, for example, nouns, verb, and adjectives.

- **Collocation Extraction:** Collocations [83] are common in natural languages, and a word cannot be classified only on the basis of its meaning, sometimes co-occurrence

with other words may alter the meaning dramatically. An example is "United States" as one term compared to the two independent terms "united" and "states", which illustrates the importance of collocations compared to single-word terms when they can be detected.

- **Word Sense Disambiguation:** The idea of word sense disambiguation (WSD) is to identify the correct sense of word (for example, two of the senses of "bank" are "river bank" and "money bank") by analyzing context within a sentence.

- **Concept Extraction:** Since a timestamp-determination task relies on statistics of words, it is difficult to determine the timestamp of a document with only a few words in common with a corpus. A possibility is to instead compare concepts in two language models in order to solve the problem of less frequent words.

- **Word Filtering:** A filtering process is needed to select the most informative words and also decrease the vocabulary size. In our work, we apply the *tf-idf* weighting scheme to each term and only the top-ranked N_t terms will be selected as representative terms for a document.

3.5 Improving Temporal Language Models

In this section, we propose three new methods for improving the JRH approach: 1) word interpolation, 2) temporal entropy, and 3) external search statistics from Google Zeitgeist [38]. Each method will be described in more details below.

3.5.1 Word Interpolation

When a word has zero probability for a time partition according to the training corpus, this does not necessarily mean the word was not used in documents outside the training corpus in that time period. It just reflects a shortcoming of having a training corpus of limited size. As described in Section 3.3.2, smoothing can be used to model that a word also exists in other time partitions.

In the following we present more elaborate ways of word frequency interpolation for partitions where a word does not occur. In this process, a word is categorized into one of two classes depending on characteristics occurring in time: *recurring* or *non-recurring*. Recurring words are words related to periodic events, for example, "French Open", "Christmas", "Olympic Games", and "World Cup", and are supposed to appear periodically in time, for example December every year, or every four years. On the other hand, non-recurring words do not appear periodically (but might still appear in many time periods, and as such can be also classified as aperiodic).

How to interpolate depends on which category a word belongs to. All words that are not recurring are non-recurring, and thus it suffices to identifying the recurring words. This can be done in a number of ways, we initially use a simple technique just looking at overlap of words distribution at endpoints of intervals, for example when detecting yearly

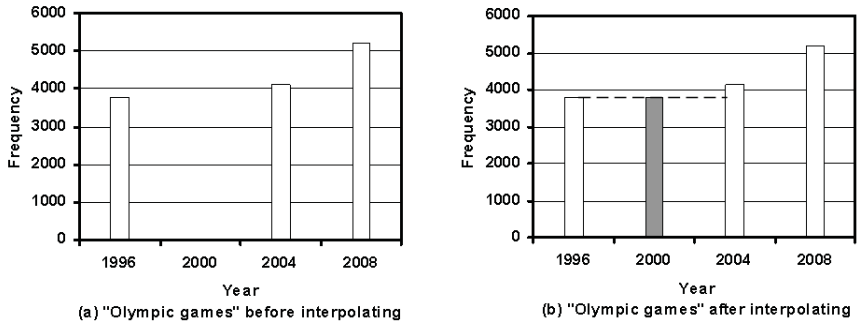


Figure 3.1: Interpolation of a recurring word.

events look at all possible 12 month intervals (i.e., words on January 2000 and January 2001, February 2000 and February 2001. Note that the endpoints should actually be a bit flexible/wide, due to the fact that many events do not occur at the exact same date each year (Easter and Olympics are two typical examples).

Our interpolation approach is based on two methods: for recurring words, if they exist in a number of event periods those that are missing are automatically "filled in", for non-recurring words interpolation is applied on periods adjacent to periods where the words exist.

Recurring Words: Assume a word w_r that has been determined to be recurring, for example "Olympic Games". If the frequency of w_r in a partition p_j, represented as $tf(w_r, p_j)$, is equal to zero, we interpolate $tf(w_r, p_j)$ with the minimum value of adjacent partitions as:

$$tf(w_r, p_j) = \min\left(tf(w_r, p_{j-1}), tf(w_r), p_{j+1}\right) \tag{3.5}$$

As depicted in Figure 3.1(a), the frequency is zero in the year 2000 (i.e., the word does not occur in any documents with timestamp within year 2000). After interpolating, Figure 3.1(b) shows how the frequency in the year 2000 is assigned with that of 1996 because it is the minimum value of 1996 and 2004.

Non-Recurring Words: Assume a word w_{nr} that has been determined to be non-recurring, for example "terrorism". Figure 3.2(a) illustrates that a frequency is missing in the year 2000 because there is no event (occurrence of word) on "terrorism" in this year. On the other hand, in the year 2001 and 2002, "terrorism" becomes popular as terrorists attacked on 11^{th} of September 2001. Once again, information about "terrorism" is absent in the year 2003. However, "terrorism" becomes popular in the year 2004 and 2005 because of bombing in Madrid and London. Supposed, there is no major event on "terrorism" after the year 2005, so the frequency is zero in the year 2006, 2007 and 2008. Although the word does not occur in the corpus it is quite certain that the word still has been used in "the real world". We interpolate $tf(w_{nr}, p_j)$ in three ways.

1. In the case of a period p_j where w_{nr} has never been seen before, it is possible to

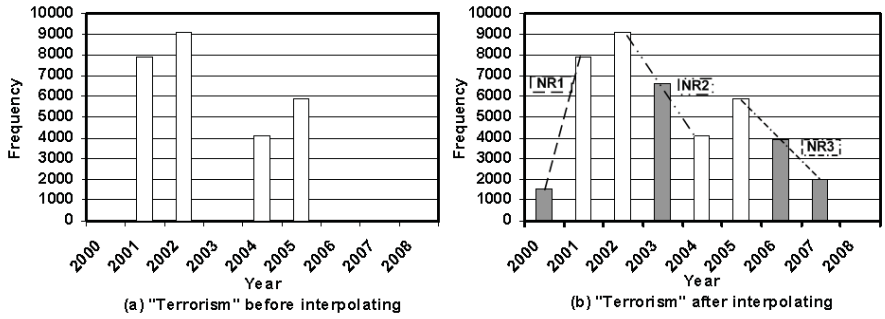


Figure 3.2: Interpolation of a non-recurring word.

observe w_{nr} in that period. We interpolate $tf(w_{nr}, p_j)$ as:

$$tf(w_{nr}, p_j) = \alpha \cdot tf(w_{nr}, p_{j+1}) \tag{3.6}$$

where α is a constant and $0 < \alpha \leq 1$. p_{j+1} is the first partition w_{nr} occurs. For example, the year 2000 is interpolated based on a fraction of the frequency in the year 2001. The interpolation method for this case is shown as *NR1* in Figure 3.2(b).

2. In the case that p_j is a period that w_{nr} is supposed to be normally used, but is absent due to missing data, we interpolate $tf(w_{nr}, p_j)$ with the average frequency of two adjacent partitions as:

$$tf(w_{nr}, p_j) = \frac{tf(w_{nr}, p_{j-1}) + tf(w_{nr}, p_{j+1})}{2} \tag{3.7}$$

For example, the year 2003 is interpolated with the average frequency of 2004 and 2005. The interpolation method of this case is shown as *NR2* in Figure 3.2(b).

3. If p_j is a period where w_{nr} is absent because of decreasing popularity of the word, it can still be expected that w_{nr} is used afterward, but not as much as before. We interpolate $tf(w_{nr}, p_j)$ as:

$$tf(w_{nr}, p_j) = \beta \cdot tf(w_{nr}, p_{j-1}) \tag{3.8}$$

where β is a constant and $0 < \beta \leq 1$. p_{j-1} is the last partition w_{nr} appears. In this case, the frequency of the years 2006, 2007 and 2008 are interpolated with a frequency of the year 2005 in a decreasing proportion. The interpolation method for this case is shown as *NR3* in Figure 3.2(b).

3.5.2 Temporal Entropy

In this section we present a term weighting scheme concerning temporality called *temporal entropy* (TE). The basic idea comes from the term selection method presented in [79]. Terms are selected based on their entropy or noise measure. Entropy of a word w_i is defined as follows:

$$Entropy(w_i) = 1 + \frac{1}{\log N_D} \sum_{d \in \mathbf{D}} P(d|w_i) \times \log P(d|w_i) \tag{3.9}$$

$$P(d_j|w_i) = \frac{tf(w_i, d_j)}{\sum_{k=1}^{N_D} tf(w_i, d_k)} \tag{3.10}$$

where N_D is the total number of documents in a collection $\mathbf{D}$ and $tf(w_i, d_j)$ is the frequency of w_i in a document d_j. It measures how well a term is suited for separating a document from other documents in a document collection, and also it captures the importance of the term within the document. A term occurring in few documents has higher entropy compared to one appearing in many documents. Therefore, the term with high entropy, is a good candidate for distinguishing a document from others.

Similar to *tf-idf* but more complicated, term entropy underline the importance of a term in the given document collection whereas *tf-idf* weights a term in a particular document only. Empirical results showing that term entropy is good for index term selection can be found in [68]. Thus, we use term entropy as a term weighting method for highlighting appropriate terms in representing a time partition.

We define temporal entropy as a measure of how well a term is suitable for separating a time partition among overall time partitions and also indicates how important a term is in a specific time partition. Temporal entropy of a term w_i is given as follows:

$$TE(w_i) = 1 + \frac{1}{\log N_P} \sum_{p \in \mathbf{P}} P(p|w_i) \times \log P(p|w_i) \tag{3.11}$$

$$P(p_j|w_i) = \frac{tf(w_i, p_j)}{\sum_{k=1}^{N_P} tf(w_i, p_k)} \tag{3.12}$$

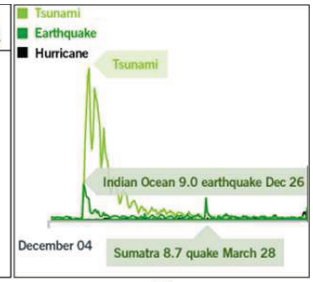
where N_P is the total number of partitions in a corpus $\mathbf{P}$, and $tf(w_i, p_j)$ is the frequency of w_i in partition p_j. Modifying the score in Equation (3.1), each term w can be weighted with temporal entropy $TE(w)$ as follows:

$$NLLR_{te}(d_i, p_j) = \sum_{w \in d_i} TE(w) \times P(w|d_i) \times \log \frac{P(w|p_j)}{P(w|C)} \tag{3.13}$$

A term that occurs in few partitions is weighted high by its temporal entropy. This results in a higher score for those partitions in which the term appears.

Top 10 Gaining Queries Week Ending Dec. 27, 2004	Top 10 Declining Queries Week Ending Dec. 27, 2004
1. tsunami	1. anna kournikova
2. santa tracker	2. jeri ellsworth
3. earthquake	3. millau bridge
4. howard hughes	4. judith regan
5. tidal wave	5. reindeer
6. reggie white	6. scott peterson
7. aleve	7. christmas
8. new year	8. nelly
9. winter solstice	9. halo 2
10. napoleon dynamite	10. heidi klum

(a)



(b)

Figure 3.3: Search statistics and trends obtained from Google Zeitgeist.

3.5.3 Search Statistics

In our work, we have also studied how to use external knowledge, and in this section we describe how to make use of search statistics provided by a search engine. The only public available statistics that suits our purpose are those from Google Zeitgeist, which is given on different time granularities, such as week, month and year. We have employed the finest granularity available, i.e., weekly data. Figure 3.3(a) shows a snapshot of search statistics which is composed of the top-10 rank for two types of queries. In the statistics, a query can be gaining or declining.

A gaining query is a keyword that is growing in interest and becomes an emerging trend at a particular time. Figure 3.3(b) shows the trend graph of the keywords "Tsunami" and "Earthquake". Both words are gaining queries in December 2004 because they gain very high frequencies compared to a normal distribution and slightly decrease their popularity over the time line. In March 2005, the word "Earthquake" becomes a gaining query again because of an earthquake in Sumatra. On the other hand, a declining query is a keyword where its interest drops noticeably from one period to another.

By analyzing search statistics, we are able to increase the probability for a particular partition which contains a top-ranked query. The higher probability the partition acquires, the more potential candidate it becomes. To give an additional score to a word w_i and a partition p_j, we check if (w_i,p_j) exist as a top-ranked query. After that, we retrieve from statistics information about a query type (gaining or declining), query ranking and the number of partitions in which w_i appears. Finally, a GZ score of w_i given p_j can be computed as:

$$GZ(p_j, w_i) = \Big(P(w_i) - f(R_{i,j}) \Big) \times ipf_i \tag{3.14}$$

where ipf_i is defined as an inverse partition frequency and is equal to $\log \frac{N_P}{n_i}$. N_P is the total number of partitions and n_i is the number of partitions containing w_i. $P(w_i)$ is the probability that w_i occurs; $P(w_i)$ = 1.0 if w_i is a gaining query word and $P(w_i)$ = 0.5 if w_i is a declining query word. This reflects the fact that a gaining query is more important than a declining one. The function $f(R_{i,j})$ takes a ranked number and converts into a weight for each word. A high ranked query is more important in this case.

We now integrate GZ as an additional score into Equation (3.1) in order to increase the probability of partition p_j:

$$NLLR_{gz}(d_i, p_j) = \sum_{w \in d_i} \left(P(w|p_j) \times \log \frac{P(w|p_j)}{P(w|C)} + \beta GZ(p_j, w) \right) \tag{3.15}$$

where β is the weight for the GZ function which is obtained from an experiment and represented by a real number between 0 and 1.

3.6 Evaluation

Our proposed enhancements were evaluated by comparing their performance in determining the timestamp with experimental results from using the JRH approach as baseline. In this section, we will describe experimental setting, experiments and results.

3.6.1 Setting

In order to assign timestamp to a document, a reference corpus consisting of documents with known dates was required for comparison. A temporal language model was then created from the reference corpus. In fact, the temporal language model is intended to capture word usage within a certain time period. Two mandatory properties of the reference corpus are:

- A reference corpus should consist of documents from various domains.
- A reference corpus has to cover the time period of a document to be dated.

We created a corpus collection from the Internet Archive [49] by downloading the history of web pages, mostly web versions of newspapers (e.g., ABC News, CNN, New York Post, etc., in total 15 sources). The corpus collection covers on average 8 years for each source and the total number of web pages is about 9000 documents, i.e., the web pages in the corpus collection have on average been retrieved once every five day by the Internet Archive crawler.

3.6.2 Experiments

In order to evaluate the performance of the enhanced temporal language models, the documents in the corpus collection were partitioned into two sets (C_{train}, C_{test}). C_{train} was used as a training set and to create a temporal language model. C_{test} was used as a testing set and to estimate timestamps of documents (note that we actually have the correct timestamps of these documents so that the precision of estimation can be calculated).

The training set C_{train} must meet the two properties mentioned above. This can be achieved by creating it based on news sources of various genres that cover the time period of documents to be dated. We chosen 10 news sources from the corpus collection to

build the training set. To create C_{test}, we randomly selected 1000 documents from the remaining 5 news sources as a testing set.

In our experiments, we used two performance measures: precision and recall. Precision in our context means the fraction of processed documents that are correctly dated, while recall indicates the fraction of correctly dated documents that are processed. A recall lower than 100% is essentially the result of using confidence of timestamping to increase precision.

The experiments were conducted in order to study three aspects: 1) semantic-based preprocessing, 2) temporal entropy (TE) and Google Zeitgeist (GZ), and 3) confidence in the timestamp-estimation task. Unfortunately, we were unable to evaluate our proposed interpolation because of a too short time span (only 8 years) in the corpus collection. However, we used linear interpolation as proposed by Kraaij [70] in our experiments, and the smoothing parameter λ is set to 0.1.

We evaluated the performance of the techniques repeating each experiment 10 times on different testing sets, which all were created based on random sampling. Averaged precision and recall were measured for each experiment.

Experiment A: In this experiment, we evaluated the performance of semantic-based preprocessing. The experiment was conducted on different combinations of semantic methods. In A.1, we studied the effect of concept extraction. C_{train} was created as a training language model with the preprocessing steps: POS tagging, WSD, concept extraction and word filtering. In A.2, we studied the effect of collocation extraction. C_{train} was created as a training language model with the preprocessing steps: POS tagging, collocation, WSD and word filtering. In A.3, C_{train} was created as a training language model with the preprocessing steps: POS tagging, collocation extraction, WSD, concept extraction and word filtering. In all experiments, timestamp was determined for documents in C_{test}. Precision was measured for each combination of semantic-based techniques.

Experiment B: In order to evaluate the performance of temporal entropy and use of Google Zeitgeist statistics, we created a training language model on C_{train} in two ways: using the semantic-based preprocessing in A.3 and without semantic-based preprocessing. For each document in C_{test} the timestamp was determined using Equations (3.13) and (3.15). Precision was measured for each scoring technique.

Experiment C: Similar to a classification task, it is necessary to know how much confidence the system has in assigning a timestamp to a document. This can for example be used as feedback to a user, or as part of a subsequent query process where we want to retrieve documents from a particular time only of the confidence of the timestamp is over a certain threshold. Confidence was measured by the distance of scores of the first and the second ranked partitions and it is given as follows.

$$\mathit{Conf}(t_{d_i}) = \log \frac{\mathit{NLLR}(d_i, p_m)}{\mathit{NLLR}(d_i, p_n)} \tag{3.16}$$

where $t_{d_i} = \mathit{PubTime}(d_i)$. p_m and p_n are the first two partitions that give the highest scores to a document d_i computed by Equation (3.1). A language model was created for C_{train} and, for each document in C_{test}, timestamp was determined by varying a confidence threshold. We measured precision and recall for each level of confidence.

Table 3.1: Results of the experiment A.

Granularities	Precision			
	Baseline	A.1	A.2	A.3
1-w	53.430	55.873	47.072	48.365
1-m	56.066	62.873	59.728	61.152
3-m	53.470	62.076	65.069	66.360
6-m	53.971	62.051	66.065	68.712
12-m	53.620	58.307	69.005	68.216

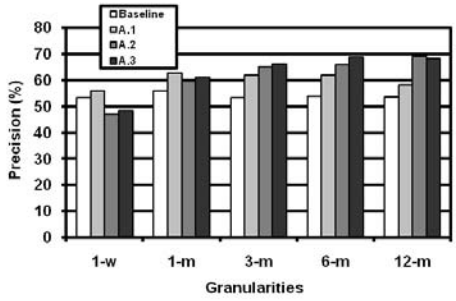
3.6.3 Results

Figure 3.4(a) (also presented in Table 3.1) presents precision of results from determining timestamp for different granularities using the baseline technique (the *JRH* approach) and combinations of different preprocessing techniques (A.1/A.2/A.3). As can be seen, by adding semantic-based preprocessing higher precision can be obtained in almost all granularities except for 1-week (where only using concept extraction outperforms the baseline). The observation indicates that using a 1-week granularity, the frequency of a collocation in each week is not so different. For example, news related to "tsunami" were reported for about 6 weeks (during December 2004 and January 2005) and each week had almost the same frequency of collocations such as "tsunami victim" and "tsunami survivor". Thus the probability of a collocation is distributed in the case of a small granularity and it is hard to gain a high accuracy for any particular partition. On the other hand, as soon as the granularity becomes more coarse, usage of collocations are quite distinct, as can be seen from the results of 1-month, 3-month, 6-month and 12-month.

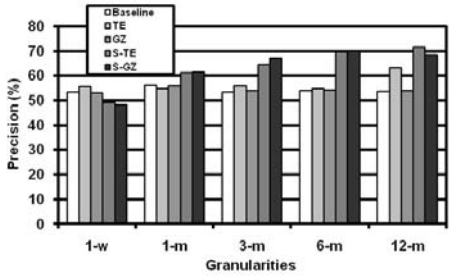
Figure 3.4(b) (also presented in Table 3.2) illustrates precision of results from determining timestamp when using temporal entropy (TE) without semantic-based preprocessing, Google Zeitgeist statistics without semantic-based preprocessing (GZ), temporal entropy with semantic-based preprocessing (S-TE), and Google Zeitgeist statistics with semantic-based preprocessing (S-GZ). As can be seen, without semantic-based preprocessing, TE only improves accuracy greatly in 12-month while in other granularities its results are not so different to those of the baseline, and GZ does not improve accuracy in all granularities. In contrast, by applying semantic-based preprocessing first, TE and GZ obtain high improvement compared to the baseline in almost all granularities except for 1-week which is too small granularity to gain high probabilities in distinguishing partitions.

From our observation, semantic-based preprocessing generates collocations as well as concept terms which are better in separating time partitions than single words. Those terms are weighted high by its temporal entropy. Similarly, most of the keywords in Google Zeitgeist statistics are noun phrases, thus collocations and concepts gains better GZ scores. This results in a high probability in determining timestamp.

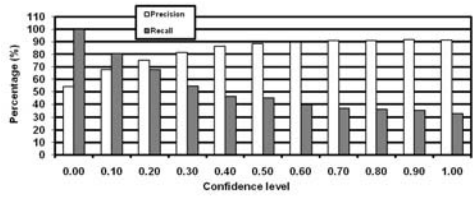
Figure 3.4(c) shows how the confidence level affects the accuracy of document dating. If the confidence level is 0, recall is 100% but precision is only 54.13%. On the other hand,



(a) Results of combining different preprocessing techniques (A.1/A.2/A.3).



(b) Results of temporal entropy and Google Zeitgeist with/without semantic-based preprocessing.



(c) Effect of a confidence level to accuracy.

Figure 3.4: Results of experiments A, B and C.

Table 3.2: Results of the experiment B.

Granularities	Precision				
	Baseline	TE	GZ	S-TE	S-GZ
1-w	53.430	55.725	53.050	49.126	48.423
1-m	56.066	54.629	56.026	61.196	61.540
3-m	53.470	55.751	54.030	64.525	67.008
6-m	53.971	54.797	54.271	69.605	69.824
12-m	53.620	63.104	53.947	71.564	68.366

if the confidence level is 1.0, precision is up to 91.35% but recall decreases to 33%. As shown in the figure, a high confidence threshold gives a high precision in determining time, whereas a document with a correctly estimated date might be discarded. Thus the confidence level can be used to provide more reliable results.

3.7 Document Dating Prototype

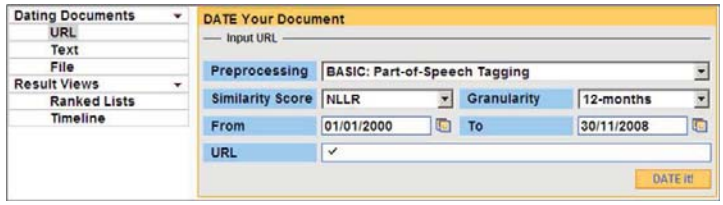
In order to demonstrate the usefulness of our research we have implemented a proof-of-concept prototype for document dating. We built the system prototype based on the proposed techniques for improving temporal language models. The prototype uses a web-based interface, and allows estimating the date of documents in different input formats (i.e. a file, contents from an URL, or text entered directly) as shown by Figure 3.5(a). Example inputs can be: 1) URL: `http://tsunami-thailand.blogspot.com`, or 2) text: the president Obama. In addition, a user can select different parameters for perform document dating.

- Preprocessing: POS, COLL, WSD, or CON
- Similarity score: NLLR, GZ or TE
- Time granularity: 1-month, 3-months, 6-months, or 12-months

Given an input to be dated, the system computes similarity scores between a given document/text and temporal language models. The document is then associated with tentative time partitions or its likely originated timestamps. As output it will present an estimation of possible creation time/periods with confidence of each of the estimated time periods, that is, a rank list of partitions ordered descendingly according to their scores as shown in Figure 3.5(b). Besides, each tentative time partition is drawn in a timeline with its score as a height as depicted in Figure 3.5(c).

3.8 Conclusions

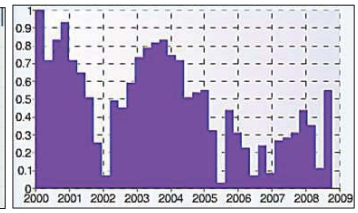
In this chapter, we have described several methods that increase the quality of determining the time of non-timestamped documents. Extensive experiments show that our approaches



(a) Input interface

Ranked ID	Date Range	Similarity Score	% Confidence
1	2000/01 - 2000/12	1.00	4.02
2	2002/12 - 2003/12	0.96	16.93
3	2003/12 - 2004/12	0.79	2.90
4	2000/12 - 2001/12	0.76	0.86
5	2007/11 - 2008/11	0.75	3.63
6	2001/12 - 2002/12	0.72	6.74
7	2004/12 - 2005/11	0.65	0.98
8	2005/11 - 2006/11	0.64	1.50
9	2006/11 - 2007/11	0.62	0.00

(b) Rank list



(c) Timeline

Figure 3.5: Input and output interfaces of the document dating system.

considerably increases quality compared to the baseline based on the previous approach by de Jong et al. Note that, although using our approach shows improvement, the quality of the actual document dating processing is still limited when aiming at further increase in effectiveness. Finally, we have presented a system prototype for document dating.

Part III

Query Analysis

Chapter 4

Determining Temporal Profiles of Queries

In the previous chapter, we proposed an approach for improving the temporal language model for determining the time of non-timestamped documents. In this chapter, we address the research question: *how to determine the time of an implicit temporal query and use the determined time for re-ranking search results?* We propose novel approaches for determining the time of a temporal query where time is not explicitly provided by a user and use the determined time for re-ranking search results.

4.1 Motivation

In previous work [12, 93], searching temporal document collections has been performed by issuing temporal queries composed of keywords, and the publication time of documents (called temporal criteria). In that way, a system narrows down search results by retrieving documents with respect to both textual and temporal criteria. As explained in Chapter 2, temporal queries can be divided into two categories: 1) those with temporal criteria explicitly provided by users [12, 93], and 2) those with no temporal criteria provided. An example of a query with temporal criteria explicitly provided is U.S. Presidential election 2008, while a query without temporal criteria provided is, for instance, Germany FIFA World Cup. However, for the latter example, a user's temporal intent is *implicitly* provided, i.e., referring to the world cup event in 2006.

More precisely, we want to determine the time of a query that is composed of *only keywords* where its relevant documents are associated to particular time periods that are not given by the query. We propose to leverage the determined time of queries, the so-called *temporal profiles* of queries, for re-ranking search results in order to increase the retrieval effectiveness. To the best of our knowledge, dating short queries and employing the determined time in ranking has not been done before.

Contributions

Our main contributions in this chapter are as follows.

- We perform the first study on how to determine the temporal profiles of queries without temporal criteria provided, and we propose techniques for determining the time of implicit temporal queries.
- We propose an approach to re-ranking search results by incorporating the determined time of queries.

Organization

In Section 4.2, we give an overview of related work. In Section 4.3, we outline the document and query models used in this chapter. In Section 4.4, we present our approaches to determining temporal profiles of queries without temporal criteria provided. In Section 4.5, we describe how to use the determined time to improve the retrieval effectiveness. In Section 4.6, we evaluate our proposed query dating, and re-ranking methods. Finally, in Section 4.7, we conclude the chapter.

4.2 Related Work

Recently, a number of papers have described issues of temporal search [12, 93, 106]. In the approaches described in [12, 93], a user explicitly specifies time as a part of query. Typically, such a temporal query is composed of query keywords and temporal criteria, which can be a point in time or a time interval. In general, temporal ranking can be divided into two types: approaches based on *link-based analysis* and *content-based analysis*. The first approach studies link structures of a document and uses this information in a ranking process, whereas the second approach examines the contents of a document instead of links. In our context, we will focus on analyzing contents only because information about links is not available in all domains, and content-based analysis seems to be more practical for a general search application. Temporal ranking exploiting document contents and temporal information are presented in [31, 51, 74, 100, 106].

In [74], Li and Croft incorporated time into language models, called time-based language models, by assigning a document prior using an exponential decay function of a document creation date. They focused on recency queries, such that the more recent documents obtain the higher probabilities of relevance. In [31], Diaz and Jones also used document creation dates to measure the distribution of retrieved documents and create the temporal profile of a query. They showed that the temporal profile together with the contents of retrieved documents can improve average precision for the query by using a set of different features for discriminating between temporal profiles. In [106], Sato et al. defined a temporal query and proposed ranking taking into account time for fresh information retrieval. In [51] an approach to rank documents by freshness and relevance is presented. In [100], Perkiö et al. introduced a process of automatically detecting a topical trend (the strength of a topic over time) within a document corpus by analyzing the temporal behavior of documents using a statistic topic model.

Berberich et al. [10] integrated temporal expressions into query-likelihood language modeling, which considers uncertainty inherent to temporal expressions in a query and documents, i.e., temporal expressions can refer to the same time interval even they are not exactly equal. The work by Berberich et al. and our work is similar in the sense that both incorporate time into a ranking in order to improve the retrieval effectiveness for temporal search, however, in their work, the temporal criteria are explicitly provided for a query. Metzler et al. [87] also consider implicit temporal needs in queries. They proposed mining query logs and analyze query frequencies over time in order to identify strongly time-related queries. In addition, they presented a ranking concerning implicit temporal needs, and the experimental results showed that their approach improved the retrieval effectiveness of temporal queries for web search. Keikha et al. [65, 66] proposed a time-based query expansion technique that selects terms for expansion from different times. Then, the technique was used for retrieving and ranking blogs, which also captures the dynamics of the topic both in aspects and vocabulary usage over time.

4.3 Models for Documents and Queries

In this section, we present models for documents and queries used in this chapter.

4.3.1 Document Model

In this chapter, a document collection contains a number of corpus documents defined as $C = \{d_1, \ldots, d_n\}$. A document d_i can be seen as bag-of-words (an unordered list of terms), and the publication time. Note that, d_i can also be associated to temporal expressions containing in the contents. However, temporal expressions will not be studied in this chapter. Let *PubTime*(d_i) be a function that gives the publication time of d_i, so d_i can be represented as $d_i = \{\{w_1, \ldots, w_n\}, \textit{PubTime}(d_i)\}$. If C is partitioned with respect to a time granularity of interest, the associated time partition of d_i is a time interval $[t_k, t_{k+1}]$ containing *PubTime*(d_i), that is *PubTime*$(d_i) \in [t_k, t_{k+1}]$. For example, if we partition C using the *1-month* granularity and *PubTime*(d_i) is 2010/03/05, the associated time partition of d_i will be $[2010/03/01, 2010/03/31]$.

4.3.2 Temporal Query Model

We define a temporal query q as composed of two parts: keywords $q_{\textit{text}}$ and temporal criteria $q_{\textit{time}}$, where $q_{\textit{text}} = \{w_1, \ldots, w_m\}$, and $q_{\textit{time}} = \{t'_1, \ldots, t'_l\}$ where t'_j is a time interval, or $t'_j = [t_j, t_{j+1}]$. In this work, we model time using a time interval only because of its simple representation. However, we note that a fine-grained time representation, such as a point in time, should also be employed in order to capture the real-world meanings of time.

In other words, q contains uncertain temporal intent that can be represented by one or more time intervals. We can refer to $q_{\textit{text}}$ as topical features and $q_{\textit{time}}$ as temporal features

of q. Hence, our aim is to retrieve documents about the topic of query where their creation dates are corresponding to time criteria.

Definition 1 (Temporal Queries)**.** *Temporal queries can be divided into two types with respect to given temporal criteria:*

(i) A query with temporal criteria explicitly provided by a user is called an explicit temporal query.

(ii) A query without temporal criteria explicitly provided is called an implicit temporal query.

An example of an explicit temporal query is Summer Olympics 2008 where a user is interested in documents about Summer Olympics written in 2008. Because we represent the time of a query by a time interval, for a given query, using the *1-year* time granularity q_{time} is represented as:

$$q_{time} = \{[2008/01/01, 2008/12/31]\}$$

Implicit temporal queries are strongly related to particular time periods although time is not given in the queries as such. An example of an implicit temporal query is Boxing Day tsunami, which is implicitly associated with the year 2004, thus q_{time} can be represented as:

$$q_{time} = \{[2004/01/01, 2004/12/31]\}$$

Another example is the query the U.S. presidential election, which can be associated with the years 2000, 2004, and 2008. So that, q_{time} is equal to:

$$q_{time} = \{[2000/01/01, 2000/12/31], \ldots, [2008/01/01, 2008/12/31]\}$$

When the time q_{time} is not given explicitly by the user, it has to be determined by the system, as will be described later in this chapter.

4.4 Determining Time of Queries using Temporal Language Models

In this section, we describe three approaches to determining the time of queries when no temporal criteria are provided. The first two approaches use the temporal language models (cf. Section 3.3) as basis, and the last approach uses no language models.

In order to build temporal language models, a temporal corpus is needed. The temporal corpus can be any document collection where 1) the documents are timestamped with creation time, 2) covering a certain time period (at least the period of the queries collections), and 3) containing enough documents to make robust models. A good basis for such a corpus is a news archive. We will use the New York Times Annotated Corpus [96] since it is readily available for research purposes. However, any corpus with

Table 4.1: Example of the contents of the temporal language models.

Time	Term	Frequency
2001	World Trade Center	1545
2002	Terrorism	2236
2003	Iraq	1510
2004	Euro 2004	750
2004	Athens	1213
2005	Terrorism	1990
2005	Tsunami	3528
2005	Hurricane Katrina	1012
2008	Obama	2030

similar characteristics can be employed, including non-English corpora for performing dating of non-English texts. We will in the following denote a temporal corpus as $\mathcal{D}_{\mathcal{N}}$.

The first approach performs dating queries using keywords only. The second approach takes into account the fact that in general queries are short, and aims at solving this problem with a technique inspired by pseudo-relevance feedback (PRF) that uses the *top-k* retrieved documents in dating queries. The third approach also uses the *top-k* retrieved documents by PRF and assumes their creation dates as temporal profiles of queries.

All approaches will return a set of determined time intervals and their weights, which will be used in re-ranking documents in order to improve the retrieval effectiveness as described in more detail in Section 4.5.

4.4.1 Dating Queries using Keywords

Our basic technique for query dating is based on using keywords only, and it is described formally in Algorithm 1.

The first step is to build temporal language models T_{LM} from the temporal document corpus (line 5), which essentially is the statistics of word usage (raw frequencies) in all time intervals, which are partitioned with respect to the selected time granularity g. Table 4.1 illustrates a subset of the temporal language models. Creating the temporal language models (basically aggregating statistics grouped on time periods) is obviously a costly process, and will be done just once as an off-line process and then only the statistics have to be retrieved at query time.

For each time partition p_j in T_{LM}, the similarity score between q_{text} and p_j is computed (line 7). The similarity score is calculated using a normalized log-likelihood ratio according to Equation 3.1. Each time partition p_j and its computed score will be stored in C, or the set of time intervals and scores (line 8). After computing the scores for all time partitions, the contents of C will be sorted by similarity score, and then the *top-m* time intervals are selected as the output set A (line 10).

Finally, the determined time intervals resulting from Algorithm 1 will be assigned weights indicating their importance. In our approach, we simply give a weight to each

Algorithm 1 *DateQueryKeywords*($q_{text}, g, m, \mathcal{D}_{\mathcal{N}}$)

1: **INPUT:** Query q_{text}, time granularity g, number of time intervals m, and temporal corpus $\mathcal{D}_{\mathcal{N}}$
2: **OUTPUT:** Set of time intervals associated to q_{text}
3: $A \leftarrow \emptyset$ *// Set of time intervals*
4: $C \leftarrow \emptyset$ *// Set of time intervals and scores*
5: $T_{LM} \leftarrow$ *BuildTemporalLM*$(g, \mathcal{D}_{\mathcal{N}})$
6: **for each** $\{p_j \in T_{LM}\}$ **do**
7: $score_{p_j} \leftarrow$ *CalSimScore*(q_{text}, p_j) *// Compute similarity score of q_{text} and p_j*
8: $C \leftarrow C \cup \{(p_j, score_{p_j})\}$ *// Store p_j and its similarity score*
9: **end for**
10: $A \leftarrow C$*.selectTopMIntervals(m)* *// Select top-m intervals ranked by scores*
11: **return** A

time interval using its reverse ranked number. For example, if the output set A contains top-5 ranked time intervals, the intervals ranked 1, 2, 3, 4, and 5 will have the weights 5, 4, 3, 2, and 1 respectively.

4.4.2 Dating Queries using Top-k Documents

In our second approach to query dating, the idea is that instead of dating query keywords q_{text} directly, we will instead date the *top-k* retrieved documents of the (non-temporal) query q_{text}. The resulting time of the query will be the combination of determined times of each top-k document.

The algorithm for dating a query using top-k retrieved documents is given in Algorithm 2. First, we retrieve documents by issuing a (non-temporal) query q_{text}, and retrieve only the *top-k* result documents (line 5). Then, temporal language models T_{LM} are built as described previously (line 6). For each document d_i in D_{TopK}, compute its similarity score with each time partition p_j in T_{LM} (lines 10-13). After computing scores for d_i for all time partitions, sort the contents of C by similarity score, and select only *top-m* time intervals as the results of d_i (line 14).

The next step is to update the set B with a set of time results C_{tmp} obtained from dating d_i. This is performed as follows: For each time interval p_k in C_{tmp}, check if B already contains p_k (line 16). If p_k exists in B, get a frequency of p_k and increase the frequency by 1 (lines 17-18). If p_k does not exist in B, add p_k into B as a new time interval and set its frequency to 1 (line 20). After dating all documents in D_{TopK}, sort the contents of B by frequency, and select only the *top-m* time intervals as the output set A (line 25).

The weights of time intervals will be their reverse ranked number. Note that it can be only one time interval in each rank of an output obtained from Algorithm 1, while it can be more than one time interval in each rank in case of Algorithm 2.

Algorithm 2 *DateQueryWithTopkDoc*($q_{text}, g, m, k, \mathcal{D}_\mathcal{N}$)

INPUT: Query q_{text} , time granularity g, number of intervals and documents m, k, temporal corpus $\mathcal{D}_\mathcal{N}$
OUTPUT: Set of time intervals associated to q_{text}
$A \leftarrow \emptyset$ *// Set of time intervals*
$B \leftarrow \emptyset$ *// Set of time intervals and their frequencies*
$D_{TopK} \leftarrow$ *RetrieveTopKDoc*(q_{text}, k) *// Retrieve top-k documents*
$T_{LM} \leftarrow$ *BuildTemporalLM*$(g, \mathcal{D}_\mathcal{N})$
for each $\{d_i \in D_{TopK}\}$ **do**
  $C \leftarrow \emptyset$ *// Set of time intervals and scores*
  $C_{tmp} \leftarrow \emptyset$ *// Set of time intervals*
  for each $\{p_j \in T_{LM}\}$ **do**
    $score_{p_j} \leftarrow$ *CalSimScore*(d_i, p_j) *// Compute similarity score of* d_i *and* p_j
    $C \leftarrow C \cup \{(p_j, score_{p_j})\}$ *// Store* p_j *and its similarity score*
  end for
  $C_{tmp} \leftarrow C$.*selectTopMIntervals(m)* *// Select top-m intervals by scores*
  for each $\{p_k \in C_{tmp}\}$ **do**
    if B *has* p_k **then**
      freq $\leftarrow B$.*getFreqForTInterval*(p_k) *// Get frequency of* p_k
      $B \leftarrow B$.*updateFreqForTInterval*$(p_k$,*freq* $+ 1)$ *// Increase frequency by 1*
    else
      $B \leftarrow B$.*addTInterval*$(p_k, 1)$ *// Add a new time interval and set its frequency to 1*
    end if
  end for
end for
$A \leftarrow B$.*selectTopMIntervals(m)* *// Select top-m intervals ranked by frequency*
return A

4.4.3 Dating Queries using Publication Time

The last approach is a variant of the dating using *top-k* documents described above. The idea is similar in the use of the *top-k* retrieved documents of the (non-temporal) query q_{text}. The resulting time of the query will be the creation date (or timestamps) of each top-k document. In this case, no temporal language models are used.

4.5 Re-ranking Documents Using Query Temporal Profiles

In this section, we will describe how to use temporal profiles of queries determined by our approaches to improve the retrieval effectiveness. The idea is that, in addition to the documents' scores with respect to keywords, we will also take into account the documents' scores with respect to the *implicit* time of queries. Intuitively, documents with creation

dates that closely match with temporal profiles of queries are more relevant and should be ranked higher.

There are a number of methods to combine a time score with existing text-based weighting models. For example, a time score can be combined with tf-idf weighting using a linear combination, or it can be integrated into language modeling using a document prior probability as in [74]. In this chapter, we propose to use a mixture model of a keyword score and a time score. Given a temporal query q with the determined time q_{time}, the score of a document d can be computed as follows:

$$S(q,d) = (1-\alpha)\cdot S'(q_{text}, d_{text}) + \alpha \cdot S''(q_{time}, d_{time}) \tag{4.1}$$

where α is a parameter underlining the importance of a keyword score $S'(q_{text}, d_{text})$ and a time score $S''(q_{time}, d_{time})$. A keyword score $S'(q_{text}, d_{text})$ can be implemented using any of existing text-based weighting models, and it can be normalized as:

$$S'_{norm}(q_{text}, d_{text}) = \frac{S'(q_{text}, d_{text})}{\max S'(q_{text}, d_{text,i})} \tag{4.2}$$

where $\max S'(q_{text}, d_{text,i})$ is the maximum keyword score among all documents.

For a time score $S''(q_{time}, d_{time})$, we formulate the probability of generating temporal profiles of query q_{time} given the associated time partition of document d_{time} as:

$$\begin{aligned} S''(q_{time}, d_{time}) &= P(q_{time}|d_{time}) \\ &= P(\{t'_1, \ldots, t'_n\}\,|d_{time}) \\ &= \frac{1}{|q_{time}|}\sum_{t'_j \in q_{time}} P(t'_j|d_{time}) \end{aligned} \tag{4.3}$$

where q_{time} is a set of time intervals $\{t'_1, \ldots, t'_n\}$, such that:

$$(t'_1 \cap t'_2 \cap \ldots \cap t'_n) = \emptyset$$

So, $P(q_{time}|d_{time})$ is an average of the probability of generating a time interval, or $P(t'_j|d_{time})$, over all the number of time intervals in q_{time}, or $|q_{time}|$.

The probability of generating a time interval t'_j given the time partition of document d_{time} can be defined in two ways as proposed in [10]: 1) ignoring uncertainty, and 2) taking uncertainty into account. By ignoring uncertainty, $P(t'_j|d_{time})$ is defined as:

$$P(t'_j|d_{time}) = \begin{cases} 0 & \text{if } d_{time} \neq t'_j, \\ 1 & \text{if } d_{time} = t'_j. \end{cases} \tag{4.4}$$

In this case, the probability of generating query time will be equal to 1 only if d_{time} is exactly the same as t'_j. By taking into account a weight of each time interval t'_j, $P(t'_j|d_{time})$ with *uncertainty-ignorant* becomes:

$$P(t'_j|d_{time}) = \begin{cases} 0 & \text{if } d_{time} \neq t'_j, \\ \frac{w(t'_j)}{\sum_{t'_k \in q_{time}} w(t'_k)} & \text{if } d_{time} = t'_j. \end{cases} \tag{4.5}$$

where $w(t'_j)$ is a function giving a weight for a time interval t'_j, which is normalized by the sum of all weights $\sum_{t'_k \in q_{time}} w(t'_k)$. In the case where uncertainty is concerned, $P(t'_j|d_{time})$ is defined using an exponential decay function:

$$P(t'_j|d_{time}) = DecayRate^{\lambda \cdot |t'_j - d_{time}|} \tag{4.6}$$

where $DecayRate$ and λ are constant, $0 < DecayRate < 1$ and $\lambda > 0$. Intuitively, this function gives a probability that decreases proportional to the difference between a time interval t'_j and the time partition of document d_{time}. A document with its time partition closer to t'_j will receive a higher probability than a document with its time partition farther from t'_j. By incorporating a weight of each time interval t'_j, $P(t'_j|d_{time})$ with *uncertainty-aware* becomes

$$P(t'_j|d_{time}) = \frac{w(t'_j)}{\sum_{t'_k \in q_{time}} w(t'_k)} \times DecayRate^{\lambda \cdot |t'_j - d_{time}|} \tag{4.7}$$

The normalization of $S''_{norm}(q_{time}, d_{time})$ are computed in two ways:

1. uncertainty-ignorant using $P(t'_j|d_{time})$ defined in Equation 4.5.
2. uncertainty-aware using $P(t'_j|d_{time})$ defined in Equation 4.7.

Finally, the normalized value of $S''_{norm}(q_{time}, d_{time})$ will be substituted $S''(q_{time}, d_{time})$ in Equation 4.1 yielding the normalized score of a document d given a temporal query q with determined time q_{time} as follows:

$$S_{norm}(q, d) = (1 - \alpha) \cdot S'_{norm}(q_{text}, d_{text}) + \alpha \cdot S''_{norm}(q_{time}, d_{time}) \tag{4.8}$$

4.6 Evaluation

In this section, we will perform two experiments in order to evaluate our proposed approaches: 1) determining temporal profiles of queries using temporal language models, and 2) re-ranking search results using the determined time. In this section, we will describe the setting for each of the experiments, and then the results.

4.6.1 Setting

As mentioned earlier, we can use any news archive collection to create temporal language models. In this chapter, we used the New York Times Annotated Corpus as the temporal corpus. This collection contains over 1.8 million articles covering a period of January 1987 to June 2007. The temporal language models were created and stored in databases using Oracle Berkeley DB version 4.7.25.

To evaluate the query dating approaches, we obtained queries from Robust2004, which is a standard test collection for the TREC Robust Track containing 250 topics (topics 301-450 and topics 601-700). As reported in [74], some TREC queries favor documents in

particular time periods. Similarly, we analyzed a distribution of relevant documents of the Robust2004 queries over time, and we randomly selected 30 strongly time-related queries (with the topic number: 302, 306, 315, 321, 324, 330, 335, 337, 340, 352, 355, 357, 404, 415, 428, 435, 439, 446, 450, 628, 648, 649, 652, 653, 656, 667, 670, 676, 683, 695). Time intervals of relevant documents were assumed as the correct time of queries.

We measured the performance using precision, recall and F-score. Precision is the fraction of determined time intervals that are correct, while recall indicates the fraction of correct time intervals that are determined. F-score is the weighted harmonic mean of precision and recall, where we set $\beta = 2$ in order to emphasize recall. For query dating parameters, we used the top-m interval with $m = 5$, and the time granularity g and the *top-k* documents were variable in the experiments.

To evaluate the re-ranking approaches, the Terrier search engine [120] was employed, and we used the BM25 probabilistic model with Generic Divergence From Randomness (DFR) weighting as our retrieval model. For the simplicity, we used default parameter settings for the weighting function. Terrier provides a mechanism to alter scores for retrieved documents by giving prior scores to the documents. In this way, we re-ranked search results at the end of retrieval by combining a keyword score $S'(q_{text}, d_{text})$ and a time score $S''(q_{time}, d_{time})$ as defined in Equation 4.8. We conducted re-ranking experiments using two collections: 1) the Robust2004 collection, and 2) the New York Times Annotated Corpus.

For the Robust2004 collection, we used the 30 queries as temporal queries without time explicitly provided. The retrieval effectiveness of temporal search using the Robust2004 collection is measured by Mean Average Precision (MAP), and R-precision. For the New York Times Annotated Corpus, we selected 24 queries from a historical collection of aggregated search queries, or the Google zeitgeist [38]. An example of temporal queries are shown in Table 4.2. The temporal searches were conducted by human judgment. Performance measures are the precision at 5, 10, and 15 documents, or P@5, P@10, and P@15 respectively. For re-ranking parameters, we used an exponential decay rate $DecayRate = 0.5$, and $\lambda = 0.5$. A mixture model parameter was obtained from the experiments, where $\alpha = 0.05$ and 0.10 for *uncertainty-ignorant* and *uncertainty-aware* methods respectively.

Table 4.2: Example of the Google zeitgeist queries and associated time intervals.

Query	Time	Query	Time
diana car crash	1997	madrid bombing	2005
world trade center	2001	pope john paul ii	2005
osama bin laden	2001	tsunami	2005
london congestion charges	2003	germany soccer world cup	2006
john kerry	2004	torino games	2006
tsa guidelines liquids	2004	subprime crisis	2007
athens olympics games	2004	obama presidential campaign	2008

The description of different approaches is given in Table 4.3. Top-k documents were retrieved using pseudo relevance feedback, i.e., the result documents after performing

query expansion using Rocchio algorithm.

Table 4.3: Different re-ranking approaches for comparison.

Method	Description
QW	determines time using keywords *plus* uncertainty-ignorant re-ranking
QW-U	determines time using keywords *plus* uncertainty-aware re-ranking
PRF	determines time using top-k retrieved documents *plus* uncertainty-ignorant re-ranking
PRF-U	determines time using top-k retrieved documents *plus* uncertainty-aware re-ranking
NLM	assumes creation dates of top-k retrieved documents as temporal profiles of queries (no language models used) *plus* uncertainty-ignorant re-ranking
NLM-U	assumes creation dates of top-k retrieved documents as temporal profiles of queries (no language models used) *plus* uncertainty-aware re-ranking

4.6.2 Results

The performance of query dating methods is shown in Table 4.4. NLM performs best in precision for all time granularities whereas PRF performs best in recall (only for *12-month*). NLM and PRF give the best F-score results for *6-month* and *12-month* respectively. In general, the smaller k tends to give the better results, while *12-month* yields higher performance compared to *6-month*. Finally, the performance of QW seems to be robust for *12-month* regardless of dating solely short keywords.

Table 4.4: Query dating performance using precision, recall and F-score.

Method	Precision		Recall		F-score($\beta = 2$)	
	6-month	*12-month*	*6-month*	*12-month*	*6-month*	*12-month*
QW	.56	.67	.34	.64	.37	.65
PRF *(k=5)*	.55	.63	.47	**.79**	.48	**.75**
PRF *(k=10)*	.56	.60	.46	.74	.48	.71
PRF *(k=15)*	.54	.60	.42	.70	.44	.68
NLM *(k=5)*	.92	**.97**	.35	.44	.40	.49
NLM *(k=10)*	.90	.95	.48	.56	.53	.61
NLM *(k=15)*	.89	.93	.56	.63	.61	.67

To evaluate re-ranking, the baseline of our experiments is a retrieval model without taking into account temporal profiles of queries, i.e., pseudo relevance feedback using Rocchio algorithm. For the Robust2004 queries, the baseline performance are MAP=0.3568 and R-precision=0.3909. Experimental results of MAP and R-precision are shown in Table 4.5. The results show that QW, QW-U, PRF and PRF-U outperformed the baseline in both MAP and R-precision for *12-month*, and NLM and NLM-U outperformed the baseline in all cases. PRF-U always performed better than PRF in both MAP and R-precision for *12-month*, while QW-U performed better than QW in R-precision for

12-month only. NLM and NLM-U always outperformed the baseline and the other proposed approaches because using the creation dates of documents is more accurate than those obtained from the dating process. This depicts that taking time into re-ranking can better the retrieval effectiveness. Hence, if query dating is improved with a high accuracy, the retrieval effectiveness will be improved significantly.

Table 4.5: Re-ranking performance using MAP and R-precision with the baseline performance 0.3568 and 0.3909 respectively (the Robust2004 collection).

Method	MAP		R-precision	
	6-month	*12-month*	*6-month*	*12-month*
QW	.3565	.3576	.3897	.3924
QW-U	.3556	.3573	.3925	.3943
PRF *(k=5)*	.3564	.3570	.3885	.3926
PRF *(k=10)*	.3568	.3570	.3913	.3919
PRF *(k=15)*	.3566	.3567	.3912	.3921
PRF-U *(k=5)*	.3548	.3574	.3903	.3950
PRF-U *(k=10)*	.3538	.3576	.3904	.3935
PRF-U *(k=15)*	.3538	.3572	.3893	.3940
NLM *(k=5)*	.3585	.3589	.3924	.3917
NLM *(k=10)*	.3586	.3591	.3918	.3925
NLM *(k=15)*	.3584	.3596	.3898	.3934
NLM-U *(k=5)*	.3604	.3608	.3975	**.3978**
NLM-U *(k=10)*	.3604	.3610	.3953	.3961
NLM-U *(k=15)*	.3606	**.3620**	.3943	.3967

Table 4.6: Re-ranking performance using P@5, P@10, and P@15 with the baseline performance 0.35, 0.30 and 0.27 respectively * indicates statistically improvement over the baselines using t-test with significant at $p < 0.05$ (the NYT collection).

Method	P@5		P@10		P@15	
	6-month	*12-month*	*6-month*	*12-month*	*6-month*	*12-month*
QW	.42	.45	.37	.39	.32	.33
QW-U	.40	.42	.35	.36	.30	.32
PRF *(k=15)*	.42	.46	.38	.42	.35	.39
PRF-U *(k=15)*	.41	.45	.36	.40	.33	.37
NLM *(k=15)*	.50	.52	.47	.49	.42	.44
NLM-U *(k=15)*	.53	**.55***	.48	**.50***	.45	**.46***

The results of evaluate the Google zeitgeist queries are shows in Table 4.6. In this case, we fix the number of *top-k* to 15 only. Table 4.6 illustrated the precision at 5, 10 and 15 documents. The baseline performance is P@5=0.35, P@10=0.30 and P@15=0.27. The results show that our proposed approaches perform better than the baseline in all cases. NLM and NLM-U performs the best among all proposed approaches.

4.7 Conclusions

In this chapter, we have studied implicit temporal queries where no temporal criteria is provided, and how to increase retrieval effectiveness for such queries. The effectiveness has been improved by determining the implicit time of the queries and employing this to re-rank the query results. Through extensive experiments we have shown that our proposed approach improves retrieval effectiveness. We note that the quality of the actual query dating processing is a limitation when aiming at further increase in the retrieval effectiveness.

Chapter 5

Handling Terminology Changes over Time

A language can change over time, which includes changes of words related to their definitions, semantics, and names (people, location, etc.). Particularly, words can be obsolete, for example, before the year 1939, the name "Siam" was used for "Thailand" and it is rarely used nowadays. This causes a problem when a user is unable to formulate a query equivalent to a term used in the collection, that is, both query and documents are represented in different forms (historical or modern forms). In this chapter, the research question we address is *how to handle terminology changes in searching temporal document collections?*

5.1 Motivation

This chapter focuses on the problem of terminology changes over time. In particular, we deal with the changes of named entities (i.e., name of people, organizations, locations, etc.) because a peculiarity of named entities compared to other vocabulary terms is that they are very dynamic in appearance, e.g., changes of roles or alterations of names. Moreover, we are interested in named entities because they constitute a major fraction of queries [18, 105]. To illustrate the problem, we give as examples two search scenarios.

First, a student studying the history of the Roman Catholic Church wants to know about the Pope Benedict XVI during the years before he became the Pope (i.e., before 2005). Using only the query Pope Benedict XVI and temporal criteria "before 2005" is not sufficient to retrieve documents about "Joseph Alois Ratzinger", which is the birth name of the current Pope. Second, a journalist wants to search for information about the past career of Hillary Rodham Clinton before becoming the 67^{th} United States Secretary of State in January 2009. When searching with the query Hillary R. Clinton and temporal criteria "before 2008", documents about "United States Senator from New York" and "First Lady of the United States" are also relevant as her roles during the years before 2008. The given examples indicate an inability of retrieving relevant documents com-

posed of the synonyms of query terms in the past. This can be considered as *semantic gaps* in searching document archives, i.e., a lack of knowledge about a query and its synonyms[1], which are semantically equivalent/related to a query with respect to time. We denote those synonyms as *time-dependent synonyms*.

This problem will be handled during query time by using a dictionary linking concepts and entities based on time, such as, by performing query expansion. Thus, for the query Thailand, the query might be expanded to Thailand or Siam. For the query Thailand and a temporal constraint before 1939, the query can be rewritten from Thailand to Siam. To improve the quality of searching historical documents by expansion, it has been done before in two manners: an expansion of query and an expansion of index. In the first case, a set of rules is automatically constructed for mapping historic terms into modern terms. In the latter case, based on a lexical database, terms are indexed together with their synonyms and holonyms as additional indices. In order to handle changing languages, we will expand a query with terms that are semantically equal with respect to temporal criteria. This we achieve by building a time-concept dictionary from the well-known and freely available encyclopedia Wikipedia.

In this chapter, we describe an approach to automatically creating entity-synonym relationships based on the contents of Wikipedia. Evolving relationships are detected using the most current version of Wikipedia, while relationships for particular time in the past are discovered through the use of snapshots of previous Wikipedia versions. In this way, we can provide a source of time-based entity-synonym relationships from 2001 until today, and using our approach also future relationships with new named entities can be discovered simply by processing Wikipedia as new contents are added. Further, we employ the New York Times Annotated Corpus in order to extend the covered time range as well as improve the accuracy of time of synonyms. Finally, we present a system prototype for searching news archives that takes into account terminology changes over time.

Contributions

Our contributions in this chapter are as follows.

- We formally model Wikipedia viewed as a temporal resource for classification of time-based synonyms.
- We propose an approach to discovering time-based synonyms using Wikipedia and improving the time of synonyms. In addition, we propose query expansion techniques that exploit time-based synonyms.
- A system prototype for searching news archives taking into account terminology changes over time is present.

[1] In general, synonyms are different words with very similar meanings. However, in this work, synonyms are words used as another name for an entity.

Organization

The organization of the rest of the chapter is as follows. In Section 5.2, we give an overview of related work. In Section 5.3, we briefly describe the assumed document model and Wikipedia features. In Section 5.4, we introduce formal models for Wikipedia viewed as a temporal resource and for time-based synonyms. In Section 5.5, we describe our approach to discovering time-based synonyms from Wikipedia. In Section 5.6, we describe how to use time-based synonyms to improve the retrieval effectiveness. In Section 5.7, we evaluate our proposed synonym detection and query expansion. In Section 5.8, we present our news search system prototype. Finally, in Section 5.9, we conclude this chapter.

5.2 Related Work

Several attempts have been made in using the semi-structured contents of Wikipedia for information retrieval purposes. The ones most relevant to our work are [18, 76, 88, 107, 134, 138]. For a thorough overview of the area of Wikipedia mining, we refer to the survey by Medelyan et al. [86].

In [138], Zesch et al. evaluate the usefulness of Wikipedia as a lexical semantic resource, and compare it to more traditional resources, such as dictionaries, thesauri and WordNet. In [18], Bunescu and Paşca study how to use Wikipedia for detecting and disambiguating named entities in open domain texts in order to improve search quality. By recognizing entities in the indexed text, and disambiguating between multiple entities sharing the same proper name, the users can access to a wider range of results as today's search engines may easily favor the most common sense of an entity, making it difficult to get a good overview of the available information for a lesser known entity.

An initial approach for synonym detection based on [18] in a non-temporal context was described in [17]. As far as we know, all previous approaches to synonym detection from Wikipedia have been based on redirects only (i.e., [48, 124, 133]) and no temporal aspects are considered. There is some work that exploits Wikipedia for query expansion. In [76], they proposed to improve the retrieval effectiveness of ad-hoc queries using a local repository of Wikipedia as an external corpus. They analyzed the categorical information in each Wikipedia article, and select terms from top-k articles to expand a query. Then, a second retrieval on the target corpus is performed. Results show that Wikipedia can improve the effectiveness of weak queries while pseudo relevance feedback is unable to improve.

Milne et al. [88] proposed an approach to help users to evolve queries interactively, and automatically expand queries with synonyms using Wikipedia. The experiments show an improvement in recall. The recent work by Xu et al. [134] tackled with a problem of pseudo-relevance feedback that one or more of the top retrieved documents may be non-relevant, which can introduce noise into the feedback process. The proposed approach in [134] classifies queries into 3 categories (entity, ambiguous, and broader queries) based on Wikipedia, and use a different query expansion method for each query category. Their experiments show that Wikipedia based pseudo-relevance feedback improves the retrieval

effectiveness, i.e., Mean Average Precision.

The affect of terminology evolution over time is addressed in [13, 28, 55, 56, 118]. We are unable to compare the performance of different methods because many of them were published at the same time or later as this work. Thus, we leave a comparison of different approaches for future work.

5.3 Preliminaries

In this section, we briefly outline models for queries and documents. In addition, we introduce temporal document collections employed in this chapter, that is, Wikipedia and the New York Time Annotated Corpus.

5.3.1 Temporal Query Model

We define a temporal query q as composed of two parts: keywords q_{text} and temporal criteria q_{time}, where $q_{text} = \{w_1, \ldots, w_m\}$, and $q_{time} = \{t'_1, \ldots, t'_l\}$ where t'_j is a time interval, or $t'_j = [t_j, t_{j+1}]$. In this work, we model time using a time interval only because of its simple representation and we aim at retrieving documents about the topic of query where their creation dates are corresponding to the time interval.

5.3.2 Document Model

In our work, we employ a *temporal document collection*, which contains documents that are temporally-ordered and it can be modeled as $C = \{d_1, \ldots, d_n\}$. A document can be seen as bag-of-words (an unordered list of terms, or features) with its associated time interval (from it was created until replaced by a new version or deleted): $d_i = \{\{w_1, w_2, w_3, \ldots, w_n\}, [t_i, t_{i+1}]\}$ where $[t_i, t_{i+1}]$ is a time interval of the document, i.e., a time period that d_i exists, and $t_i < t_{i+1}$. $\mathit{PubTime}(d_i)$ is a function that gives the publication date of the document and must be valid within the time interval, and $\mathit{PubTime}(d_i) \in [t_i, t_{i+1}]$.

5.3.3 Temporal Document Collections

Generally, temporal document collections are document collections where their contents appear in a temporal order, such as, web archives, news archives, blogs, personal emails and enterprise documents. In such domains, terms in the text streams are temporally dynamic in pattern, e.g., rising sharply in frequency, growing in intensity for a period of time, and then fading away. In the following, we present two temporal document collections that are used in this chapter.

Wikipedia

Wikipedia is a freely available source of knowledge. Each editable article in Wikipedia has associated revisions, i.e., all previous versions of its contents. Each revision (or a

version) of an article is also associated with a time period that it was in use before being replaced by the succeeding version. In other words, the time of a revision is a time period when it was a current version.

There are four Wikipedia features that are particularly attractive as a mining source when building a large collection of named entities: article links (internal links in one Wikipedia article to another article), redirect pages (send a reader to another article), disambiguation pages [2] (used by Wikipedia to resolve conflicts between terms having multiple senses by either listing all the senses for which articles exist), and categories (used to group one or more articles together, and every article should preferably be a member of at least one category although this is not enforced).

New York Time Annotated Corpus

The New York Times Annotated Corpus is used in the synonym time improvement task. This collection contains over 1.8 million articles covering a period of January 1987 to June 2007. 1.5 million articles are manually tagged of vocabulary of people, organizations and locations using a controlled vocabulary that is applied consistently across the collections. For instance, if one article mentions "Bill Clinton" and another refers to "President William Jefferson Clinton", both articles will be tagged with "CLINTON, BILL". Some statistics of tagged documents are given in Table 5.1.

Table 5.1: NYT collection statistics of tagged vocabulary.

Tagged Vocabulary	#Documents Tagged
People	1,328,045 (71.6%)
Locations	600,114 (32.3%)
Organizations	596,890 (32.2%)

5.4 Temporal Models of Wikipedia

In this section, we will present temporal models of Wikipedia, i.e., synonym snapshots. The models will be later used for detecting synonyms over time. Finally, we will give a formal definition of four different classes of synonyms, and how to classify them using temporal patterns of occurrence as a feature.

5.4.1 Synonym Snapshots

In our context, a document collection is Wikipedia $\mathcal{W}$ that consists of a set of articles or pages, $\mathcal{P} = \{p_1, \ldots, p_n\}$. A page $p_i \in \mathcal{P}$ consists of a set of terms and a time interval: $p_i = \{\{w_1, \ldots, w_n\}, [t_a, t_b]\}$, where $w_i \in \mathcal{V}$ and $\mathcal{V}$ is the complete set of terms or a

[2] Note that the meaning of the term *disambiguation* in Wikipedia context is slightly different from how it is used in computational linguistics.

vocabulary in the collection. A time interval $[t_a, t_b]$ is a time period that p_i exists in the collection. Wikipedia pages $\mathcal{P}$ can be categorized into two types: those that describe a named entity, e.g., a concept about people, companies, organizations, etc., and those not referring to a named entity, e.g., user talk pages and category pages.

We call a page in the first type *a named entity page*. For simplicity, we will use the term "entity" and "named entity" interchangeably. A named entity obtained from Wikipedia can be defined as:

Definition 2 (Named Entity)**.** *A named entity e_i is represented by terms constituting the title of an entity page p_e that can be obtained using the function $Entity(p_e)$.*

Let x be any object, e.g., a page p_i, or a named entity e_i. We define *TInterval*(x) as a function that gives a time interval associated to x, i.e., a time period of existence $[t_y, t_z]$. We define *TStart*(x) as a function that gives the starting time point of x, i.e., the smallest time point t_y from the time interval $[t_y, t_z]$ of x, and *TEnd*(x) as a function that gives the ending time point of x, i.e., the largest time point t_z from the time interval $[t_y, t_z]$ of x.

A page p_i is associated to a set of its revisions $\{r_j | r_j \in \mathcal{R}_i\}$. A revision r_j consists of two components: 1) a set of terms $\{w_1, \ldots, w_m\}$, and 2) a time interval $[t_c, t_d)$, which can be obtained as *TInterval*(r_j). Thus, a revision $r_j = \{\{w_1, \ldots, w_m\}, [t_c, t_d)\}$. Note that a time interval of any r_j excludes its last time point, $[t_c, t_d) = [t_c, t_d] - \{t_d\}$. Let $\mathcal{R}_i$ is a set of revisions $\{r_1, \ldots, r_n\}$ of a page p_i. The time interval of a revision $r_j \in \mathcal{R}_i$ overlaps with the time interval of p_i, that is, *TInterval*$(r_j) \subset$ *TInterval*(p_i).

The intersection of the time intervals of all revisions in $\mathcal{R}_i$ can be computed as:

Definition 3 (Intersection of Revisions)**.** *The intersection of the time of all revisions in $\mathcal{R}_i$ is an empty set. It is because at any time point t in TInterval(p_i), only one revision r_j can exist for p_i, that is:*

$$\textit{TInterval}(r_1) \cap \textit{TInterval}(r_2) \cap \ldots \cap \textit{TInterval}(r_{n-1}) \cap \textit{TInterval}(r_n) = \emptyset \qquad (5.1)$$

Time intervals of two adjacent revisions can be defined in term of each other as the follows.

Definition 4 (Two Adjacent Revisions)**.** *Let r_j and r_{j+1} be any two adjacent revisions, we can define the time intervals of these two revisions as:*

1. *TInterval*$(r_j) = [$*TStart*$(r_j),$ *TStart*$(r_{j+1}))$
2. *TInterval*$(r_{j+1}) = [$*TEnd*$(r_j),$ *TEnd*$(r_{j+1}))$

By partitioning $\mathcal{W}$ with respect to a time granularity g, we will have a set of snapshots of Wikipedia $\mathbb{W} = \{W_{t_1}, \ldots, W_{t_z}\}$. In our work, we only use the *1-month* granularity. Hence, if we have the history of Wikipedia for 8 years and $g = month$, the number of snapshots will be $|\mathbb{W}| = 8 * 12 = 96$, i.e., $\mathbb{W} = \{W_{03/2001}, \ldots, W_{03/2009}\}$. A Wikipedia snapshot can be defined as:

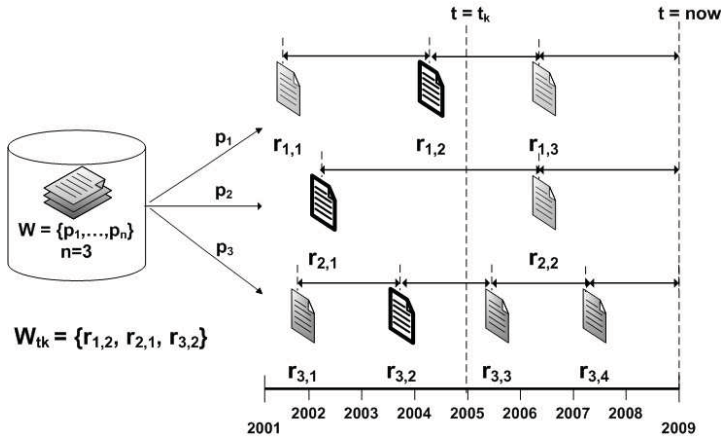


Figure 5.1: Wikipedia snapshot at time t_k and its current revisions.

Definition 5 (Wikipedia Snapshot)**.** *A snapshot W_{t_k} consists of the current revision r_c of every page p_i at time t_k:*

$$W_{t_k} = \{r_c | \forall p_i : r_c \in \mathcal{R}_i \wedge t_k \in \mathit{TInterval}(r_c) \wedge \cap \mathit{TInterval}(r_c) \neq \emptyset\} \tag{5.2}$$

Because all revisions are current at time t_k, the intersection of the time intervals of all revisions in W_{t_k} is not an empty set. Figure 5.1 depicts a snapshot W_{t_k} of Wikipedia and current revisions at time $t = t_k$.

Let $\mathcal{S}$ be a set of synonyms $\{s_1, \ldots, s_m\}$ of all entities in $\mathcal{W}$, where each synonym $s_j \in \mathcal{V}$. An entity e_i is associated to a set of synonyms $\{s_1, \ldots, s_u\}$. An entity-synonym relationship can be defined as:

Definition 6 (Entity-synonym Relationship)**.** *We define an entity-synonym relationship $\xi_{i,j}$ is a pair of an entity e_i and its associated synonym s_j, that is:*

$$\xi_{i,j} = (e_i, s_j) \tag{5.3}$$

Instead of referring to a synonym s_j alone, we must always refer to an entity-synonym relationship $\xi_{i,j}$, because s_j can be a synonym of one or more entities. An entity-synonym relationship $\xi_{i,j}$ has an associated time interval $[t_\alpha, t_\beta]$, i.e., a time period that s_j is a synonym of e_i.

The time points t_α and t_β can be obtained using *TInterval*($\xi_{i,j}$), *TStart*($\xi_{i,j}$), and *TEnd*($\xi_{i,j}$) respectively. We define a synonym snapshot as:

Definition 7 (Synonym Snapshot)**.** *A synonym snapshot S_{t_k} is defined as a set of entity-synonym relationships at a particular time $t = t_k$, that is:*

$$S_{t_k} = \{\xi_{1,1}, \ldots, \xi_{n,m}\}, t_k \in \mathit{TInterval}(\xi_{i,j}) \tag{5.4}$$

5.4.2 Time-based Classes of Synonyms

In this section, we give the definition of time-based classes of synonyms. The intuition behind the synonyms classes is that, synonyms occur differently over time, so they should be employed differently as well. Consequently, we will classify synonyms into different classes based on their occurrence patterns over time.

Let t_α^w be the starting time point and t_β^w be the last time point of the document collection, i.e., Wikipedia. Hence, $t_\alpha^w = \mathit{TStart}(\mathcal{W})$ and $t_\beta^w = \mathit{TEnd}(\mathcal{W})$. For every entity-synonym relationship $\xi_{i,j}$, let $t_\alpha^{\xi_{i,j}}$ be the first time point we observe $\xi_{i,j}$ and $t_\beta^{\xi_{i,j}}$ be the last time point we observe $\xi_{i,j}$, so $t_\alpha^{\xi_{i,j}} = \mathit{TStart}(\xi_{i,j})$ and $t_\beta^{\xi_{i,j}} = \mathit{TEnd}(\xi_{i,j})$. Figure 5.2 depicts occurrence patterns of different synonym classes over time.

The first class of synonyms is called **time-independent**, and it is defined as:

Definition 8 (Time-independent Synonyms)**.** *An entity-synonym relationship $\xi_{i,j}$ is classified as "time-independent" (Class A) if all of the following conditions hold:*

(i) $t_\alpha^{\xi_{i,j}} \in [t_\alpha^w, t_\alpha^w + \delta_1]$ *where* $\delta_1 > 0$

(ii) $t_\beta^{\xi_{i,j}} = t_\beta^w$

The idea of Class A is to detect synonyms that exist for a long time interval, as long as that of Wikipedia. These synonyms are robust to change over time and can represent good candidates of synonyms. For example, the synonym "Barack Hussein Obama II" is a time-independent synonym of the entity "Barack Obama". We use δ_1 to relax a condition of starting time because there are not many pages created at the beginning of Wikipedia. For example, δ_1 can be 24 months after Wikipedia was created.

The second class of synonyms is called **time-dependent**, and it is defined as:

Definition 9 (Time-dependent Synonyms)**.** *An entity-synonym relationship $\xi_{i,j}$ is classified as "time-dependent" (Class B) if all of the following conditions hold:*

(i) $t_\alpha^{\xi_{i,j}}, t_\beta^{\xi_{i,j}} \in \left[t_\alpha^w + \delta_1, t_\beta^w - \delta_2\right]$ *where* $\delta_2 > 0, t_\alpha^{\xi_{i,j}} > t_\beta^{\xi_{i,j}}$

(ii) $\lambda_1 \leq t_\beta^{\xi_{i,j}} - t_\alpha^{\xi_{i,j}} \leq \lambda_2$ *where* $\lambda_1, \lambda_2 > 0, \lambda_2 > \lambda_1$

The idea of Class B is to detect synonyms that are highly related to time, for example, "Cardinal Joseph Ratzinger" is a synonym of "Pope Benedict XVI" before 2005. We interest in using this synonym class for query expansion to handle the effect of rapidly changing synonyms over time as explained in Section 5.1. δ_2 indicates that synonyms are no longer in use, and it can be 12 months. λ_1, λ_2 represents minimum, maximum values of a time interval of synonym respectively. For example, λ_1 and λ_2 can be 2 months and 24 months. If a time interval is less than 2 months, it is a noise or junk synonym, and if it is greater than 24 months, it is less specific to time.

In addition to Class A and B, we observe some synonyms cannot be classified into the two classes above because of their temporal characteristics. Thus, we introduce two fuzzy-membership classes, and the first class called **gaining synonymy** is defined as:

Definition 10 (Gaining Synonyms)**.** *An entity-synonym relationship $\xi_{i,j}$ is classified as "gaining synonymy" (Class C) if all of the following conditions hold:*

(i) $t_{\alpha}^{\xi_{i,j}} \in [t_{\alpha}^{w} + \delta_1, t_{\alpha}^{w} + \delta_1 + \epsilon]$ *where* $\epsilon > 0$

(ii) $t_{\beta}^{\xi_{i,j}} = t_{\beta}^{w}$

The idea of Class C is to detect synonyms that exist for a long time interval, *but not* as long as that of Wikipedia. These synonyms can be considered good candidates of synonyms as they are tentative to robust to change over time. However, it is *not* confident to judge if they are time-independent or not. This class of synonyms is actually a special type of Class A that lacks of data in early years. For example, the synonym "Pope" has occurred as a synonym of the entity "Pope Benedict XVI" in 04/2005. Hence, this synonym will be classified to Class C instead of Class A because of its time interval. ϵ is a parameter for the missing data of early years, e.g., ϵ can be 24 months.

The final fuzzy-membership class called **declining synonymy** is defined as:

Definition 11 (Declining Synonyms)**.** *An entity-synonym relationship $\xi_{i,j}$ is classified as "declining synonymy" (Class D) if all of the following conditions hold:*

(i) $t_{\alpha}^{\xi_{i,j}} \in [t_{\alpha}^{w}, t_{\alpha}^{w} + \delta_1]$

(ii) $t_{\beta}^{\xi_{i,j}} \in \left[t_{\beta}^{w} - \theta - \delta_2, t_{\beta}^{w} - \delta_2\right]$ *where* $\theta > 0$

The idea of Class D is to detect synonyms that are stopped using as synonyms for some time ago, i.e., not in use at the moment. We can consider this class of synonym as *out-of-date* synonyms. For example, for the entity "Bill Clinton", the synonym "President Clinton" is less popular nowadays and it is very rare to be used. Thus, this synonym will belong to Class D. Synonyms in this class can be viewed as a special type of Class B. They are equivalent to synonyms in the past, but their time intervals are not too specific to particular time, i.e., greater than a certain period of time. The period of time is determined by θ that can be 12 months.

5.5 Time-based Synonym Detection

In this section, we will present our approach to find time-based entity-synonym relationships. The approach is divided into three main steps: 1) named entity recognition and synonym extractions, 2) improving time of synonyms using a model for temporal dynamics of text streams, and 3) synonym classification.

5.5.1 Named Entity Recognition and Synonym Extraction

First, we partition the Wikipedia collection according to the time granularity $g = month$ in order to obtain a set of Wikipedia snapshots $\mathbb{W} = \{W_{t_1}, \ldots, W_{t_z}\}$.

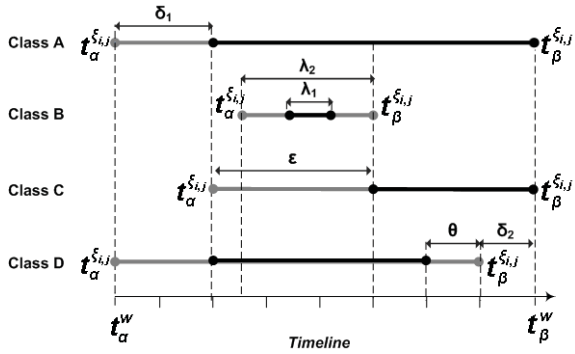


Figure 5.2: Temporal patterns of time-based classes of synonyms.

For each Wikipedia snapshot W_{t_k}, we identify all entities in a snapshot W_{t_k}. A result from this step will be a set of entities E_{t_k} at a particular time t_k. After that, we determine a set of synonyms for each entity $e_i \in E_{t_k}$ in this snapshot W_{t_k}. A result from this process is a set of entity-synonym relations, that is a synonym snapshot $S_{t_k} = \{\xi_{1,1}, \ldots, \xi_{n,m}\}$. We repeat this process for every Wikipedia snapshot W_{t_k} in $\mathbb{W}$. The final result will be the union of all synonym snapshots $\mathbb{S} = \{S_{t_1} \cup \ldots \cup S_{t_z}\}$. $\mathbb{S}$ will be input of the time-based synonym classification step.

Step 1: Recognizing named entities. Given a Wikipedia snapshot W_{t_k}, we have a set of pages existing at time t_k, that is $W_{t_k} = \{p_i | \forall p_i : t_k \in \mathit{TInterval}(p_i)\}$. In this step, we only interest in an entity page p_e. In order to identify an entity page, we use the approach described by Bunescu and Paşca in [18] which is based on the following heuristics:

- If multi-word title with all words capitalized, except prepositions, determiners, conjunctions, relative pronouns or negations, consider it an entity.
- If the title is a single word, with multiple capital letters, consider it an entity.
- If at least 75% of the occurrences of the title in the article text itself are capitalized, consider it an entity.

After identifying an entity page p_e from a snapshot W_{t_k}, we will have a set of entity pages $\mathcal{P}_{e,t_k} = \{p_e | p_e \in W_{t_k}\}$. From this set, we will create a set of entities E_{t_k} at time t_k by simply extracting a title from each entity page $p_e \in \mathcal{P}_{e,t_k}$. A result from this step is a set of entities $E_{t_k} = \{e_1, \ldots, e_n\}$, which will be used in step 2.

Step 2: Extracting synonyms. After identifying a set of entities E_{t_k}, we want to find synonyms for each entity $e_i \in E_{t_k}$. Owing to its richness of semantics structure, it is possible to use article links and redirect pages in Wikipedia for finding synonyms. However, we will not use redirect pages in this chapter because it is problematic to define a temporal model of redirect pages. Hence, we will find synonyms by extracting anchor texts from article links. For a page $p_i \in W_{t_k}$, we list all internal links in p_i but only those links that point to an entity page $p_e \in \mathcal{P}_{e,t_k}$ are interesting. In other words, the system

extracts as synonyms all anchor texts for the associated entity, and these synonyms are weighted by their frequencies of occurrence. We then obtain a set of entity-synonym relationships. By accumulating a set of entity-synonym relationships from every page $p_i \in W_{t_k}$, we will have a set of entity-synonym relationships at time t_k, i.e., a synonym snapshot $S_{t_k} = \{\xi_{1,1}, \ldots, \xi_{n,m}\}$.

Step 1 and 2 are processed for every snapshot $W_{t_k} \in \mathbb{W}$. Finally, we will obtain a set of entity-synonym relationships from all snapshots $\mathbb{S} = \{S_{t_1}, \ldots, S_{t_z}\}$, and a set of synonyms for all entities $\mathcal{S} = \{s_1, \ldots, s_y\}$. Table 5.2 depicts examples of entity-synonym relationships and their time periods extracted from Wikipedia. Note that, time periods of some relationships in Table 5.2 are incorrect. For example, the synonym "Cardinal Joseph Ratzinger" of the entity "Pope Benedict XVI" should associates with a time period before 2005. Consequently, in order to improve time periods, the results from this step will be input to the next subsection.

Table 5.2: Entity-synonym relationships and time periods.

Named Entity	Synonym	Time Period
Pope Benedict XVI	**Cardinal Joseph Ratzinger**	**05/2005 - 03/2009**
	Cardinal Ratzinger	05/2005 - 03/2009
	Joseph Cardinal Ratzinger	05/2005 - 03/2009
	Joseph Ratzinger	05/2005 - 03/2009
	Pope Benedict XVI	05/2005 - 03/2009
Barack Obama	Barack Hussein Obama II	02/2007 - 03/2009
	Barack Obama	02/2007 - 03/2009
	Obama	04/2006 - 03/2009
	Sen. Barack Obama	07/2007 - 03/2009
	Senator Barack Obama	05/2006 - 03/2009
Hillary Rodham Clinton	Hillary Clinton	08/2003 - 03/2009
	Hillary Rodham	10/2002 - 03/2009
	Hillary	07/2004 - 03/2009
	Mrs. Clinton	07/2005 - 03/2009
	Sen. Hillary Clinton	03/2007 - 03/2009
	Senator Clinton	11/2007 - 03/2009

5.5.2 Improving Time of Entity-synonym Relationships

The time periods of entity-synonym relationships do not always have the desired accuracy. The main reason for this is that the Wikipedia history has a very short timespan of only 8 years. That is, the time periods of synonyms are timestamps of Wikipedia articles in which they appear, not the time extracted from the contents of Wikipedia articles. Consequently, the maximum timespan of synonyms has been limited by the time of Wikipedia. In order

to discover the more accurate time, we need to analyze a document corpus with the longer time period, i.e., the New York Time Annotated Corpus.

There are a number of methods for extracting the more accurate time of synonyms. The easiest method is to find the starting time and the ending time, or the first point and the last point in the corpus, at which a synonym is observed with its frequency greater than a threshold. However, the problems with this method are that:

1. It cannot deal with sparse/noisy data.

2. It cannot find multiple, discontinuous time intervals of a synonym.

Alternatively, we can apply the method called "burst detection", proposed in [67] for detecting the time periods of synonyms from the corpus. Bursts are defined as points where a frequency of term increases sharply, and the frequency may oscillate above and below the threshold, resulting in a single long interval of burst or a sequence of shorter ones. Consequently, burst periods can formally represent periods that synonyms are "in use" over time.

The advantage of this method is that it is formally modeled and capable of handling sparse/noisy data. In addition, it can identify multiple, discontinuous time intervals for all terms in the document corpus. Readers can refer to Chapter 2 for detailed description of the algorithm for burst detection.

We propose to improve the time period of each entity-synonym relationship $\xi_{i,j} \in \mathbb{S}$ by analyzing the NYT corpus (with the longer timespan of 20 years) using the burst detection algorithm. The process of detecting entity-synonym relationships from the NYT corpus is as follows. First, we have to identify a synonym s_j from document streams. Note the difference between an entity-synonym relationship $\xi_{i,j}$ and a synonym s_j, the first one refers to a tuple of synonym s_j and its associated named entity e_i, while the latter one refers to a synonym s_j only.

Second, we have to find a named entity e_i associated to the identified synonym s_j because s_j can be a synonym of more than one named entity. We call this process *synonym disambiguation*. Finally, after we disambiguate synonyms, we will then obtain bursty periods of each entity-synonym relationship $\xi_{i,j}$ that can be represented more accurate time periods of $\xi_{i,j}$.

Identifying and Disambiguating Synonyms using the NYT corpus

To identify a synonym s_j from the text streams of the NYT corpus is not straightforward, because a synonym s_j can be ambiguous (i.e., a synonym may be associated with more than one named entities as Table 5.3 shows the number of synonyms associated with the different number of named entities). For example, there are more than 19,000 synonyms associating with more than one named entities, while 2.5 million synonyms associate with only one named entities. In order to disambiguate a named entity e_i for a synonym s_j, we can make use of a controlled vocabulary of the NYT corpus described in Section 5.3.

Recall that input of this step is a set of all synonyms of all entities $\mathcal{S}$ obtained from Subsection 5.5.1. The algorithm for identifying a synonym s_j from the text streams is

Table 5.3: Synonyms and corresponding named entities.

#Named Entity	#Synonym
1	2,524,170
2	14,356
3	2,797
4	994
5	442
6	259
7	155
8	94
9	58
10	37

given in Algorithm 3 and Algorithm 4. An explanation is as follows. Algorithm 3 finds a synonym s_j from each document d_n where s_j can have the maximum size of n-grams of, or w called the window size of synonym. In this case, a synonym that its size is greater than w is not interesting. Table 5.4 shows synonyms with different n-grams.

Table 5.4: Examples of Synonyms with different n-grams.

N-gram	Synonym
2	Jospeh Ratzinger
3	Senator Barack Obama
5	George III of Great Britain
6	United Nations Commission on Human Rights
8	Society for the Prevention of Cruelty to Animals
13	Queen Elizabeth II of the United Kingdom of Great Britain and Northern Ireland

First, read a term s_j with the maximum size w from a document d_n starting at the index pointer $ptr = 0$ as in Algorithm 4 (line 7). Check whether s_j is a synonym ($s_j \in \mathcal{S}$), and retrieve all associated named entities for s_j as in Algorithm 4 (line 9). Next, check if s_j has only one associated named entity, then s_j is not ambiguous, as in Algorithm 4 (line 10-11). If s_j is associated with more than one named entities, disambiguate its named entities as in Algorithm 4 (line 13-15). After disambiguating the named entities for s_j, insert an entity-synonym relationship (e_i, s_j) plus the publication time of d_n, i.e., $\mathit{PubTime}(d_n)$, in the output set and move the index pointer by the size of s_j, that is $ptr = (ptr + w)$ in Algorithm 3(line 11-12).

If s_j cannot be disambiguated, s_j will be ignored and we continue identifying another synonym, i.e., reading a term with the maximum size w from d_n by increasing the index pointer to the next word $ptr = (ptr + 1)$ as in Algorithm 3 (line 14). On the contrary, if a term s_j is not a synonym ($s_j \notin \mathcal{S}$), decrease a window size by 1 as in Algorithm 3 (line 20), and consider a prefix string of s_j with a size of $(w - 1)$, or s_{j+1}. If s_{j+1}

is not a synonym, repeat the same process until a window size w is equal to 0 as in Algorithm 4 (line 4). This means, if no any prefix substring of s_j has been recognized as a synonym, continue to read the next term with the maximum size w from the text streams by increasing the index pointer to the next word $ptr = (ptr + 1)$ as in Algorithm 3 (line 14).

Algorithm 3 *IdentifyEntitySynonymInNYT($\mathcal{D}_\mathcal{N}$)*

1: **INPUT:** $\mathcal{D}_\mathcal{N}$ is a set of documents in the NYT corpus.
2: **OUTPUT:** A sequence of $\xi_{i,j}$ or (e_i, s_j) and its timestamp.
3: $C \leftarrow \emptyset$ *// A set of entity-synonyms relationships and a time point.*
4: **for each** $\{d_n \in \mathcal{D}_\mathcal{N}\}$ **do**
5: $len_d \leftarrow |d_n|$ *// len_d is the number of words in d_n.*
6: $ptr \leftarrow 0$ *// ptr is an index pointer in d_n, default is 0.*
7: $w \leftarrow c$ *// w is the window size of synonym, default is c.*
8: **while** $ptr \leq len_d$ **do**
9: $(e_i, s_j) \leftarrow$ *FindSynonym*(d_n, ptr, w)
10: **if** $(e_i, s_j) \neq$ *null* **then**
11: $C \leftarrow C \cup \{(e_i, s_j), Time(d_n)\}$ *// Output (e_i, s_j) and publication time of d_n*
12: $ptr \leftarrow (ptr +$ *CountWords*$(s_j))$ *// Move ptr by the number of words in s_j.*
13: **else**
14: $ptr \leftarrow (ptr + 1)$ *// Move ptr to the next word.*
15: **end if**
16: **end while**
17: **end for**
18: **return** C

After identifying s_j as a synonym, it is necessary to determine whether s_j is ambiguous or not. Note that we retrieve the set of all entities E_j associated with s_j as in Algorithm 4 (line 9). If there is only one entity in E_j, s_j is not ambiguous and that entity will be assigned to s_j as in Algorithm 4 (line 10-11). However, if there are more than one entity, s_j have to be disambiguated by using controlled vocabulary V_n tagged in the document d_n as in Algorithm 4 (line 13).

The algorithm for disambiguating named entities for a synonym is given in Algorithm 5. For each entity $e_k \in E_j$, if e_k is in a set of tagged vocabulary V_n of d_n, add e_k into a list of disambiguated entities E_{tmp} as in Algorithm 5 (line 7-8). Continue for all entities in E_k. If E_{tmp} contains only one entity, s_j is disambiguated. If E_{tmp} has more than one entity, s_j cannot be disambiguated.

The final results will be tuples of disambiguated entity-synonym relationships associated with timestamps of documents where they occur. Table 5.5 illustrates results from this step of the synonyms "President Reagan" and "Senator Clinton" of the named entities "Ronald Reagan" and "Hillary Rodham Clinton" respectively. Each tuple is composed of an entity-synonym relationship, the publication time of a document where it occurs, and its frequency. Note that, one entity-synonym relationship can be associated to *different* timestamps. This is equivalent to the statistics of a entity-synonym relationship over time

Algorithm 4 *FindSynonym(d_n, ptr, w)*

INPUT: A document d_n, a pointer ptr, a size of synonym w.
OUTPUT: An entity-synonym relationship (e_i, s_j) or $\xi_{i,j}$.
$(e_i, s_j) \leftarrow$ *null* // *Set a tuple result to null.*
if $w = 0$ **then**
  return (e_i, s_j)
else
  $s_j \leftarrow$ *ReadString*(d_n, ptr, w) // *Read s_j from d_n at index ptr.*
  if $s_j \in \mathcal{S}$ **then**
    $E_j \leftarrow$ *GetAssocEntities*(s_j) // *All entities associated to s_j.*
    if $|E_j| = 1$ **then**
      $e_i \leftarrow E_j$*.firstElement()*
    else
      $e_k \leftarrow$ *Disambiguate*(d_n, E_j) // *Disambiguate E_j.*
      if $e_k \neq$ *null* **then**
        $e_i \leftarrow e_k$
      end if
    end if
    return (e_i, s_j)
  else
    FindSynonym$(d_n, ptr, (w-1))$ // *Find a synonym with a size $(w-1)$.*
  end if
end if

extracted from text streams of documents. The results from this step will be input to the next subsection.

Improving Time of Synonyms using Burst Detection

In this step, we will find the correct time of a entity-synonym relationship $\xi_{i,j}$ by using the burst detection algorithm described in [67]. The algorithm takes the results from the previous step as input, and generates bursty periods of $\xi_{i,j}$ by computing a rate of occurrence from document streams. An output produced in this step is bursty intervals and bursty weight, which are corresponding to periods of occurrence and the intensity of occurrence respectively, as showed in Table 5.6.

Detected bursty periods are mostly composed of discontinuous intervals because the algorithm depends heavily on a frequency of $\xi_{i,j}$ in the text streams. A gap in time intervals prevents us from classifying $\xi_{i,j}$ as time-independent since a time-independent synonym should have a long and continuous time interval. A solution to this problem is to combine two adjacent intervals and interpolate their bursty weight. However, interpolation for $\xi_{i,j}$ will be performed only if a synonym of $\xi_{i,j}$ has no other candidate named entities according to the fact that the relationship of a named entity and its synonym can change over time. A result from this step is a set of entity-synonym relationships, that is $\mathcal{S} = \{\xi_{1,1}, \ldots, \xi_{n,m}\}$ and more accurate time.

Algorithm 5 *Disambiguate(d_n, E_j)*

INPUT: A document d_n, and a set of associated entities E_j.
OUTPUT: A disambiguated entity.
$E_{tmp} \leftarrow \emptyset$ *// A temporary list of entities.*
$e_i \leftarrow$ *null* *// An output entity.*
$V_n \leftarrow$ *GetVocabulary*(d_n) *// Tagged vocabulary of* d_n.
for each $e_k \in E_j$ **do**
  if $e_k \in V_n$ **then**
    $E_{tmp} \leftarrow E_{tmp} \cup \{e_k\}$
  end if
end for
if $|E_{tmp}| = 1$ **then**
  $e_i \leftarrow E_{tmp}$*.firstElement()*
end if
return e_i

Table 5.5: Tuples of entity-synonym relationships.

Timestamp	Entity	Synonym	Frequency
01/1987	President Reagan	Ronald Reagan	54
03/1987	President Reagan	Ronald Reagan	23
11/1988	President Reagan	Ronald Reagan	11
01/1989	President Reagan	Ronald Reagan	34
10/1990	President Reagan	Ronald Reagan	12
04/2001	Senator Clinton	Hillary Rodham Clinton	67
05/2002	Senator Clinton	Hillary Rodham Clinton	121
05/2003	Senator Clinton	Hillary Rodham Clinton	33
11/2004	Senator Clinton	Hillary Rodham Clinton	61
01/2005	Senator Clinton	Hillary Rodham Clinton	359

5.5.3 Time-based Synonym Classification

To classify an entity-synonym relationship $\xi_{i,j}$ based on time is straightforward. The starting time point $t_{\alpha}^{\xi_{i,j}}$ and the ending time point $t_{\beta}^{\xi_{i,j}}$ of $\xi_{i,j}$ will be used to determine synonym classes as defined in Subsection 5.4.2. In this work, we are only interested in using time-independent and time-dependent synonyms for query expansion because synonyms from the other two classes might not be useful in this task. In the next section, we will explain how can we actually make use of time-based synonyms in improving the retrieval effectiveness.

Table 5.6: Results from burst-detection algorithm.

Synonym	Entity	Burst Weight	Time	
			Start	End
President Reagan	Ronald Reagan	5506.858	01/1987	02/1989
President Ronald	Ronald Reagan	100.401	01/1989	03/1990
President Ronald	Ronald Reagan	67.208	07/1990	02/1993
Senator Clinton	Hillary Rodham Clinton	18.214	01/2001	10/2001
Senator Clinton	Hillary Rodham Clinton	17.732	05/2002	01/2003
Senator Clinton	Hillary Rodham Clinton	172.356	06/2003	11/2004

5.6 Query Expansion

In this section, we will describe how to use time-based synonyms (time-independent and time-dependent synonyms) to improve the retrieval effectiveness. The use of synonyms will be divided into two different search scenarios.

The first scenario is to use time-independent class of synonyms in an ordinary search, for example, searching with keywords only (no temporal criteria explicitly provided). The usefulness of time-independent synonyms is that they can be viewed as good candidate synonyms for a named entity. For example, the synonym "Barack Hussein Obama II" is better than "Senator Barack Obama" as a synonym for the named entity "Barack Obama" in this case. Consequently, a query containing named entities can be expanded with their time-independent synonyms before performing a search.

Another case is when performing a temporal search, we must take into account *changes in semantics*. For example, searching documents about "Pope Benedict XVI" written "before 2005", documents written about "Joseph Alois Ratzinger" should also be considered as relevant because it is a synonym of the named entity "Pope Benedict XVI" at the years "before 2005". In this case, a time-dependent synonym with respect to temporal criteria can be used to expand a query before searching.

In the rest of this section, we will describe how we actually expand a query with time-based synonyms.

5.6.1 Using Time-independent Synonyms

Before expanding a query and performing an ordinary search, synonyms must be ranked according to their weights. We define a weighting function of time-independent synonyms as a mixture model of a temporal feature and a frequency feature as follows:

$$\mathit{TIDP}(s_j) = \mu \cdot pf(s_j) + (1 - \mu) \cdot \overline{tf}(s_j) \tag{5.5}$$

where $pf(s_j)$ is a time partition frequency or the number of time partitions (or time snapshots) in which a synonym s_j occurs. $\overline{tf}(s_j)$ is an averaged term frequency of s_j in all

time partitions:

$$\overline{tf}(s_j) = \frac{\sum_i tf(s_j, p_i)}{pf(s_j)} \tag{5.6}$$

where μ underlines the importance of a temporal feature and a frequency feature. In our experiments, 0.5 is a good value for μ.

Intuitively, this function measures how popular synonyms are over time. The popularity of synonym over time is measured using two factors. First, synonyms should be robust to change over time as defined in 5.4.2. Hence, the more partitions synonyms occur, the more robust to time they are. Second, synonyms should have high usages over time. This corresponds to having a high value of averaged frequencies over time.

We intend to use time-independent synonyms in order to improve the effectiveness of an ordinary search, i.e., search without temporal criteria. In this chapter, we will perform an ordinary search using Terrier search engine developed by University of Glasgow.

Given a query q, first we have to identify a named entity in query. Note that, we could not rely on state-of-the-art named entity recognition because queries are usually very short (i.e., 2-3 words on average), and lacked of standard form, e.g., all words are lower case. In addition, we need to identify a named entity corresponding to a title of Wikipedia article since our named entities and synonyms are extracted from Wikipedia.

We do this by searching Wikipedia with a query q, and q is a named entity if its search result exactly matches with a Wikipedia page. Besides, a more relax method is to select the top-k related Wikipedia pages instead. Now, we obtain a set of named entities $E_q = \{e_{q,1}, \ldots, e_{q,n}\}$ of q. Subsequently, time-independent synonyms of q are all synonyms corresponding to a named entity $e_{q,i} \in E_q$. Next, we will rank those synonyms by their *TIDP* scores and select only top-k synonyms with highest scores for expansion. Query expansion of time-independent synonyms can be performed in three ways as follows:

1. Add the top-k synonyms to an original query q, and search.

2. Add the top-k synonyms to an original query q, and search with pseudo relevance feedback.

3. Add the top-k synonyms to an original query q plus *TIDP* scores as boosting weight, and search with pseudo relevance feedback.

Boosting weight is a weight of term as defined in Terrier's query language. Note that, if synonyms are duplicated with an original query q, we will remain the original query q unchanged, and add those duplicated synonyms with *TIDP* scores as boosting weight.

5.6.2 Using Time-dependent Synonyms

In order to rank time-dependent synonyms, we first have obtain a set of synonyms from time t_k and weight them differently according to the following weighting function.

$$TDP(s_j, t_k) = tf(s_j, t_k) \tag{5.7}$$

where $tf(s_j, t_k)$ is a term frequency of a synonym s_j at time t_k. Note that, a time partition frequency is not counted because synonyms from the same time period should be equal with respect to time. Thus, only a term frequency will be used to measure the importance of synonym. Time-dependent synonyms will be used for a temporal search, or a search taking into account a temporal dimension, i.e. extending keyword search with the publication time of documents. In that way, a search system will retrieve documents according to both textual and temporal criteria, e.g., *temporal text-containment search* [93].

Given a temporal query (q, t_k), we will recognize named entities in a query q using the same method as explained in Section 5.6.1. After obtaining a set of named entities $E_q = \{e_{q,1}, \ldots, e_{q,n}\}$ of a query q, we will perform two steps of filtering synonyms. First, only synonyms which their time overlaps with time t_k will be processed, that is:

$$\{s_j | \mathit{Time}(s_j) \cap t_k \neq \emptyset\}$$

Second, those synonyms will be ranked by their *TDP* scores and select only top-k synonyms with highest scores for expansion. Using time-dependent synonyms in a temporal search is straightforward. A set of synonyms will be add into an original temporal query (q, t_k). In the following subsection, we will explain how to automatically generate temporal queries that will be later used in temporal search experiments.

5.7 Evaluation

In this section, we will evaluate our proposed approaches (extracting and improving time of synonyms, and query expansion using time-based synonyms). Our experimental evaluation is divided into three main parts: 1) extracting entity-synonym relationships from Wikipedia, and improving time of synonyms using the NYT corpus, 2) query expansion using *time-independent synonyms*, and 3) query expansion using *time-dependent* synonyms. In this section, we will describe the setting for each of the main experiments, and then the results.

5.7.1 Setting

We will now describe in detail the experimental setting of each of the experiments.

Extracting and Improving Time of Synonyms

To extract synonyms from Wikipedia, we downloaded the complete dump of English Wikipedia from the Internet Archive [129]. The dump contains all pages and revisions from 03/2001 to 03/2008 in XML format, and the decompressed size is approximately 2.8 Terabytes. A snapshot was created for every month resulting in 85 snapshots (03/2001, 04/2001, . . ., 03/2008). In addition, we obtained 4 more snapshots (05/2008, 07/2008, 10/2008, 03/2009), where 2 of them were downloaded [130]. So, we have 89 (85+4) snapshots in total.

We used the tool called MWDumper [91] to extract pages from the dump file, and stored the pages and revisions of 89 snapshots in databases using Oracle Berkeley DB version 4.7.25. We then created temporal models of Wikipedia from all of these snapshots.

To improve time of synonyms, we used the burst detection algorithm implemented by the author in [67] and the NYT corpus described in Section 5.3.3. An advantage of this implementation is that no preprocessing is performed on the documents. Parameter for burst detection algorithm were set as follows: the number of states was 2, the ratio of rate of second state to base state was 2, the ratio of rate of each subsequent state to previous state (for states > 2) was 2, and gamma parameter of the HMM was 1. We use accuracy to measure the performance of our method for improving time of synonyms.

Query Expansion using Time-independent Synonyms

To perform an ordinary search, the experiments were carried out using the Terrier search engine. Terrier provides different retrieval models, such as divergence from randomness models, probabilistic models, and language models. In our experiments, documents were retrieved for a given query by the BM25 probabilistic model with Generic Divergence From Randomness (DFR) weighting. In addition, it provides flexible query language that allows us to specify a boosting weight for a term in query. Given an initial query q_{org}, an expanded query q_{exp} with top-k synonyms $\{s_1, \ldots, s_k\}$ plus *TIDP* scores as boosting weight can be represented in Terrier's query language as follows.

$$q_{exp} = q_{org} \;\; s_1 \,\hat{}\, w_1 \;\; s_2 \,\hat{}\, w_2 \;\; \ldots \;\; s_k \,\hat{}\, w_k$$

where w_k is a time-independent weight of a synonym s_k, and computed using the function *TIDP*(s_k) defined in Equation 5.5.

We conducted an ordinary search using the standard Text Retrieval Conference (TREC) collection Robust2004. Robust2004 is the test collection for the TREC Robust Track containing 250 topics (topics 301-450 and topics 601-700). The Robust2004 collection statistics are given in Table 5.8. The retrieval effectiveness of query expansion using time-independent of synonyms is measured by Mean Average Precision (MAP), R-precision and recall. Recall in our experiments is the fraction of relevant documents Terrier retrieves and all relevant documents for a test query.

Query Expansion using Time-dependent Synonyms

To perform a temporal search, we must identify temporal queries used for a search task. We do this in an automatic way by detecting named entities that can represent temporal queries for performing temporal search experiments. Thus, named entities of interesting should have many *time-dependent* synonyms associated to them. To automatically generate temporal queries is composed of two steps as follows.

Given entity-synonym relationships $\mathbb{S} = \{\xi_{1,1}, \ldots, \xi_{n,m}\}$. First, we find temporal query candidates by searching for any named entity e_i which the number of its synonyms is greater than a threshold φ. Nevertheless, in this case, most of synonyms may be *time-independent*, and named entities become less appropriate to represent temporal queries.

Then, we must take into account a *TIDP* of each synonym. The intuition is that the lower *TIDP* weight a synonym has, the better time-dependent it is. So, named entities with an average of *TIDP* weight less than a threshold ϕ probably associate with many *time-dependent* synonyms. This makes them good candidate for temporal queries. In our experiment, the threshold of the number of synonyms φ and a threshold of the average of *TIDP* weight ϕ are 30 and 0.2 respectively.

Table 5.7: Examples of temporal queries and synonyms.

Temporal Query		Synonym
Named Entity	Time Period	
American Broadcasting Company	1995-2000	Disney/ABC
Barack Obama	2005-2007	Senator Obama
Eminem	1999-2004	Slim Shady
Eminem	2000-2002	Marshall Mathers
George H. W. Bush	1988-1992	President George H.W. Bush
George H. W. Bush	2000-2003	George Bush Sr.
George W. Bush	2000-2007	President George W. Bush
George W. Bush	2002-2005	Bush 43
Hillary Rodham Clinton	2001-2007	Senator Clinton
Kmart	1987-1992	Kmart Corporation
Kmart	1987-1987	Kresge
Pope Benedict XVI	1988-2005	Cardinal Ratzinger
Ronald Reagan	1987-1989	Reagan Revolution
Ronald Reagan	1987-1989	President Reagan
Rudy Giuliani	1994-2001	Mayor Rudolph Giuliani
Tony Blair	1998-2007	Prime Minister Tony Blair
Virgin Media	1999-2002	Telewest Communications

The temporal searches were conducted by human judgment using 3 users. Some examples of temporal queries are shown in Table 5.7. Each tuple contains a temporal query (a named entity and time criteria), and its synonym with respect to time criteria. We performed a temporal search by submitting a temporal query to the news archive search engine [92]. We compared the results of top-k retrieved documents of each query *without* synonym expansion, and those of the same query *with* synonym expansion. A retrieved document can be either *relevant* or *irrelevant* with respect to temporal criteria. According to the lacking of a standard test set (with all relevant judgments available), we could not evaluate temporal search using recall as we intended. Thus, performance measures are the precision at 10, 20 and 30 documents, or P@10, P@20, and P@30 respectively.

5.7.2 Results

First, we will show the results of extracting synonyms, and improving time of synonyms. Then, the results of query expansion using *time-independent* synonyms and the results of

Table 5.8: Robust2004 collection statistics.

Document Collection	#Docs	Size (GB)	Time Period
Financial Times	210,158	0.56	1991-1994
Federal Register	55,630	0.40	1994
FBIS	130,471	0.47	1996
Los Angeles Times	131,896	0.48	1989-1990
All	528,155	1.9	1989-1994, 1996

query expansion using *time-dependent* synonyms will be presented respectively.

Extracting and Improving Time of Synonyms

Different named entity recognition methods is described in Table 5.9. Note that, *filtering criteria for synonyms* of BPF-NERW are including: 1) the number of time intervals is less than 6 months, and 2) the average frequency (the sum of frequencies over all intervals divided by the number of intervals) is less than 2. The filtering aims to remove noise synonyms. For BPC-NERW, uninteresting categories are those none of "people", "organization" or "company".

Table 5.9: Different named entity recognition methods.

NER Method	Description
BP-NERW	Bunescu and Paşca's named entity recognition of Wikipedia (cf. Section 5.5.1)
BPF-NERW	BP-NERW with *filtering criteria for synonyms*
BPC-NERW	BP-NERW filtered out named entities in uninteresting categories
BPCF-NERW	BPC-NERW with *filtering criteria for synonyms*

The statistics obtained from extracting synonyms from Wikipedia are in Table 5.10.

Table 5.10: Statistics of entity-synonym relationships extracted from Wikipedia.

NER Method	#NE	#NE-Syn.	Max. Syn. per NE	Avg. Syn. per NE
BP-NERW	2,574,319	7,820,412	631	3.0
BPF-NERW	2,574,319	3,199,115	162	1.2
BPC-NERW	473,829	1,503,142	564	3.2
BPCF-NERW	473,829	488,383	148	1.0

In Table 5.10, Columns 2-3 are the total number of named entities recognized, and the total number of entity-synonym relationships extracted from Wikipedia, respectively. Column 4 is the maximum number of synonyms per named entity. Column 5 is the average number of synonyms per named entity.

The results from improving time of synonyms using the NYT corpus are in Table 5.11. Note that, only entity-synonym relationships without noise synonyms are interesting, i.e., recognized by the methods BPF-NERW and BPCF-NERW. In Table 5.11, Column 2 is the number of entity-synonym relationships that can be identified and assigned time from the NYT corpus using the method in Section 5.5.2. The percentage of the number of entity-synonym relationships identified and assigned time is shown in Column 3.

In order to evaluate the accuracy of the method for improving time of entity-synonym relationships, we randomly selected 500 entity-synonym relationships and manually assessed the accuracy of time periods assigned to those entity-synonym relationships. The accuracy of the method for improving time of entity-synonym relationships is shown in Column 4. The accuracy of the method for improving time of entity-synonym relationships in a case of BPCF-NERW is better than that of BPF-NERW because named entities recognized by BPF-NERW is too generic, and it is rare to gain high frequencies in the NYT corpus.

Table 5.11: Accuracy of improving time using the NYT corpus.

NER Method	#NE-Syn. Disambiguated	Accuracy (%)
BPF-NERW	393,491 (12.3%)	51
BPCF-NERW	73,257 (15.0%)	73

Query Expansion using Time-independent Synonyms

We evaluate our proposed query expansion by comparing different methods described in Table 5.12. Note that, Pseudo relevance feedback was performed by selecting 40 terms from top-10 retrieved documents, and those expansion terms were weighted by DFR term weighting model, i.e., Bose-Einstein 1.

Table 5.12: Different query expansion methods for comparison.

Method	Description
PM	(Baseline1) the probabilistic model without query expansion
RQ	(Baseline2) query expanding by re-weighting the original query
PRF	(Baseline3) query expanding by pseudo relevance feedback (Rocchio algorithm)
SQE	(Approach1) add the top-k synonyms to an original query before search
SQE-PRF	(Approach2) add the top-k synonyms to an original query and search with PRF
SWQE-PRF	(Approach3) add the top-k synonyms to an original query plus their *TIDP* scores as boosting weight, and search with PRF

Test queries were selected from the Robust2004 test set using named entities in a query described in Section 5.6.1. Note the difference between Bunescu and Paşca's named entity recognition for Wikipedia page (BP-NERW), and named entity recognition in a query (NERQ). The first method recognizes whether a Wikipedia document is a named

entity or not, and it needs to analyze the content of the Wikipedia document. For the second method, we have only a set of short queries (without a document) and we need to identify named entities in those queries. Recall that there are two methods for recognizing named entities in queries described in Section 5.6.1: 1) exactly matched Wikipedia page (MW-NERQ), and 2) exactly matched Wikipedia page and top-k related Wikipedia pages (MRW-NERQ). We used $k = 2$ in our experiments because k greater than 2 can introduce noise to the NERQ process.

The number of queries from the Robust2004 test set recognized using two methods are shown in Table 5.13. There are total 250 queries from Robust2004. MW-NERQ can recognize 42 named entity queries while MRW-NERQ can recognize 149 named entity queries. Note that, 42 and 149 queries are the number of queries found as Wikipedia article, and recognized as named entities. For example, there are actually 58 queries from Robust2004 found as Wikipedia article, but only 42 are *named entity* queries.

Table 5.13: Number of queries using two different NER.

Type	MW-NERQ	MRW-NERQ
Named entity	42	149
Not named entity	208	101
Total	250	250

Named-entity queries recognized using two NER methods are shown in Table 5.14. Each row represents different retrieval results of each retrieval method, and two main column represents two different methods for NERQ. Different retrieval results are composed of Mean Average Precision (MAP), R-precision and recall. As seen in Table 5.14, our proposed query expansion methods SQE-PRF and SWQE-PRF performs better than the baselines PM, RQ and PRF in both MAP and recall for MW-NERQ. However, there is only SWQE-PRF outperforming the baselines in R-precision. Also note that, SQE-PRF has better recall than SWQE-PRF, while the opposite seems to hold for precision. In the case of MRW-NERQ, our proposed query expansion methods have really worse performance than in the case of MW-NERQ due to the accuracy of the recognition method.

Query Expansion using Time-dependent Synonyms

The baseline of our experiments is to search using a temporal query (TQ), i.e., a keyword w_q and time t_q. Our propose method is to expand an original query with synonyms with respect to time t_q and search (TSQ). Experimental results of P@10, P@20 and P@30 of *20* of temporal query topics are shown in Table 5.15. The results show that our query expansion using time-dependent synonyms TSQ performed significantly better than temporal searches without expansion TQ. Our observation is that TQ retrieved zero to a few relevant documents (less than 10) for most of temporal queries, while TSQ could retrieve more relevant documents as a result of expanding temporal queries with time-dependent synonyms.

Table 5.14: Performance comparisons using MAP, R-precision, and recall for named entity queries, * indicates statistically improvement over the baselines using t-test with significant at $p < 0.05$.

Method	MW-NERQ			MRW-NERQ		
	MAP	R-precision	Recall	MAP	R-precision	Recall
PM	0.2889	0.3309	0.6185	0.2455	0.2904	0.5629
RQ	0.2951	0.3266	0.6294	0.2531	0.2912	0.5749
PRF	0.3469	0.3711	0.6944	**0.3002**	**0.3227**	**0.6761**
SQE	0.3046	0.3360	0.6574	0.2123	0.2499	0.5385
SWQE	0.3054	0.3399	0.6475	0.2399	0.2820	0.5735
SQE-PRF	0.3608*	0.3652	**0.7405***	0.2507	0.2665	0.5932
SWQE-PRF	**0.3653***	**0.3861***	0.7388*	0.2885	0.3080	0.6504

Table 5.15: Performance comparisons using P@10, P@20 and P@30 for temporal queries * indicates statistically improvement over the baseline using t-test with significant at $p < 0.05$.

Method	P@10	P@20	P@30
TQ	0.1000	0.0500	0.0333
TSQ	**0.5200***	**0.3800***	**0.2800***

5.8 News Archives Search System Prototype

In this section, we present a system prototype for search news archives that takes into account terminology changes over time. Our system consists of two parts: 1) the offline module for extracting time-based synonyms by using our proposed approach, as depicted in Figure 5.3, and 2) the online module for searching news archive as illustrated in Figure 5.4. With a web-based interface, the system can take as input a named entity query. It automatically determines time-based synonyms for a given named entity, and ranks the synonyms by their time-based scores. Then, a user can expand the named entity with the synonyms in order to improve the retrieval effectiveness.

Consider an example of search as also illustrated in Figure 5.4. A student studying the history of the Roman Catholic Church wants to know about the Pope Benedict XVI during the years before he became the Pope (i.e. before 2005). The student searches using the query Pope Benedict XVI and the publication dates 1987/01 and 2005/04. The system retrieves documents for the query Pope Benedict XVI, and also determines synonyms for the query with respect to time criteria. The student then selects the synonyms "Cardinal Joseph Ratzinger" to expand the query. The new query becomes Pope Benedict XVI *OR* Cardinal Joseph Ratzinger. He performs search again, and the system retrieves documents which are relevant to both "Pope Benedict XVI" and "Cardinal Joseph Ratzinger".

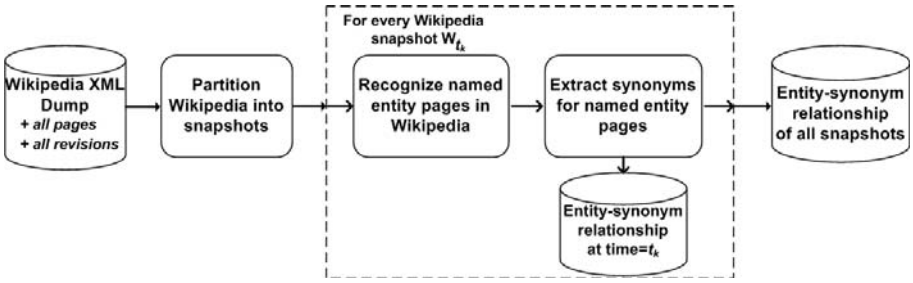


Figure 5.3: System architecture of the module for extracting time-based synonyms.

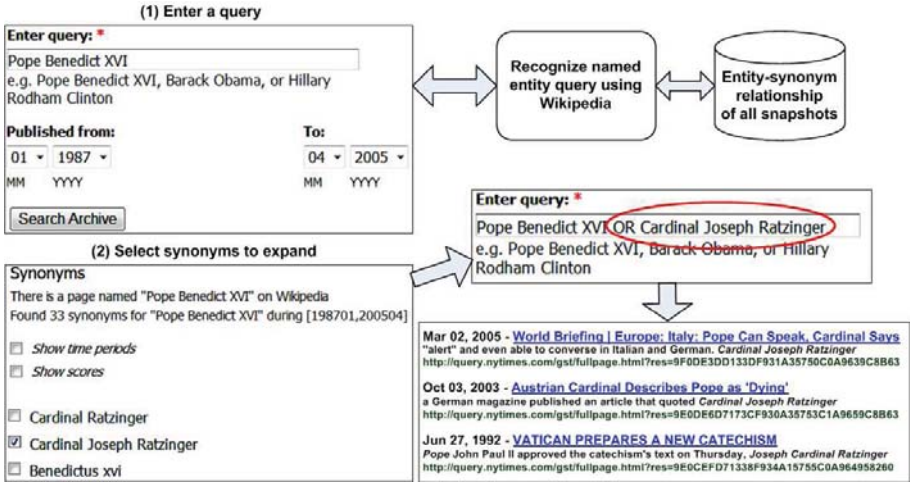


Figure 5.4: User interface of the news archives search system prototype.

5.9 Conclusions

In this chapter, we have described how to use Wikipedia to discover time-dependent and time-independent synonyms. These classified synonyms can be employed in a number of application areas, and in this chapter we have described how to perform query expansion using the time-based synonyms. The usefulness of this approach has been demonstrated through an extensive evaluation, which have showed significant increase in retrieval effectiveness. Finally, we presented a system prototype for searching news archives taking into account terminology changes over time.

Chapter 6

Time-based Query Performance Predictors

Query performance prediction is aimed at predicting the retrieval effectiveness that a query will achieve with respect to a particular ranking model. In this chapter, we study query performance prediction for temporal queries when the time dimension is explicitly modeled into ranking. This chapter addresses the research question *how to predict the retrieval effectiveness of temporal queries?*

6.1 Motivation

Retrieval effectiveness can be increased by employing *pseudo-relevance feedback* (PRF), which can be done in two steps. First, the initial search is performed for a given query, where a set of top-k retrieved documents are assumed to be relevant. Second, terms are extracted from those top-k documents and the query is automatically expanded with extracted terms for performing the second search that delivers the final results. For temporal search, *time-based pseudo-relevance feedback* (T-PRF) proposed in Chapter 4 can be employed, where the time of initial top-k retrieved documents are assumed to be relevant and the query is automatically expanded with the relevant time before the second search. However, the performance of using PRF and T-PRF depends on the quality of the initial results: with less relevant documents expanding the query can lead to query drift and possibly lower retrieval effectiveness. In that case, the search system should instead help the user to manually reformulate the query by performing *query suggestion* of *terms* and/or *time* relevant to the query, for example, giving a list of all volcanic mountains and time periods of major eruptions in Iceland for the query given above.

In this chapter, we aim at improving retrieval effectiveness for temporal search by studying *temporal query performance prediction*, i.e., predicting the retrieval effectiveness that temporal queries will achieve with respect to a particular ranking model in advance of, or during the retrieval stage in order that particular actions can be taken to improve the overall performance. In other words, query performance prediction can be

useful in order to choose between alternative query enhancement techniques described above, such as, query expansion and query suggestion. To the best of our knowledge, query performance prediction for temporal search has never been done before.

Contributions

The main contributions of this chapter are:

- We perform the first study and analysis of performance prediction methods for temporal queries.
- We propose different time-based predictors and techniques for improving query performance prediction by combining multiple predictors.

Organization

The organization of the rest of the chapter is as follows. In Section 6.2, we give an overview of related work. In Section 6.3, we first outline models for time, queries and documents. Then, we explain a temporal ranking method and define the problem of temporal query performance prediction. In Section 6.4, we present existing predictors proposed in previous work. In Section 6.5, we propose different time-based predictors and explain methods for combining different predictors. In Section 6.6, we describe how to combine different prediction in order to improve predicting performance using two methods: linear regression and neural networks. In Section 6.7, we evaluate different single predictors and the combined methods. Then, we discuss the results and conclude our findings. Finally, in Section 6.8, we summarize our work in this chapter.

6.2 Related Work

The problem of query performance prediction has recently gained a lot of attention [25, 31, 44, 42, 45, 46, 47, 108, 123, 143, 144]. Different approaches to predicting query performance can be categorized according to two aspects [44]: 1) time of predicting (pre/post-retrieval) and 2) an objective of task (difficulty, query rank, effectiveness). Pre-retrieval based approaches predict query performance independently from a ranking method and the ranked list of results. Typically, pre-retrieval based methods are preferred to post-retrieval based methods because they are based solely on query terms, the collection statistics and possibly external sources, e.g., WordNet or Wikipedia. On the contrary, post-retrieval based approaches are dependent on the ranked list of results. Pre-retrieval predictors can be classified into four different categories based on the predictor taxonomy defined by Hauff et al. [42]: 1) specificity, 2) ambiguity, 3) ranking sensitivity, and 4) term relatedness.

The first group of pre-retrieval predictors estimates the effectiveness of a query by measuring the specificity of query terms and assumes that the more specific a query, the better it will perform. In order to determine the specificity, different heuristics are proposed, for example, the averaged length of a query *AvQL* [89], the averaged inverse document frequency *AvIDF* [25] and the averaged inverse collection term frequency *AvICTF* [46].

The summed collection query similarity *SumSCQ* [143] employs both term frequencies and inverse document frequencies. In addition, the simplified (pre-retrieval) version of Clarity Score [46] (denoted *SCS*). *SCS* measures the maximum likelihood of query terms to determine the specificity. Note that, the Clarity Score is a post-retrieval method and originally proposed in [25].

An ambiguity-based predictor is for example a set coherence score [47] measuring the ambiguity of a query by calculating the similarity between all documents that contain the query term. Unfortunately, this predictor is computationally expensive although it is suggested in [47] that a subset of documents used for computing can be randomly selected. Although pre-retrieval predictors do not perform actual retrieval, we can still predict query performance by examining *term weights* of queries (e.g. tf-idf) that will be used for ranking documents. This will help in estimating how easy/hard it is for a retrieval system to rank documents containing query terms. An example of *ranking-sensitivity* based methods is the sum of query weight deviation *SumVAR* [143].

The predictors presented previously ignore *term relatedness* among query terms. Consider the queries wright brothers and right brothers, we can expect that the first one is the easiest query because wright and brothers have a stronger relationship than right and brothers. To measure the relationship between two terms, pointwise mutual information (PMI) can be computed [42]. PMI measures the relationship by observing co-occurrence statistics of terms in a document collection. Two PMI-based predictors are including the averaged PMI value of all query term pairs *AvPMI* [42] and the maximum PMI value of all query term pairs *MaxPMI* [42]. We will refer to the predictors presented above as *keyword-based* predictors.

A number of ranking models exploiting temporal information have been proposed for example [10, 31]. In [31], Diaz and Jones measure the distribution of creation dates of retrieved documents to create the temporal profile of a query, and use the profile to predict precision. Note that, we have temporal information needs explicitly provided, so we do not need to estimate a temporal profile. Berberich et al. [10] integrated temporal expressions into query-likelihood language modeling, which considers uncertainty inherent to temporal expressions in a query and documents, i.e., temporal expressions can refer to the same time interval even they are not exactly equal.

6.3 Problem Definition

We describe the models for queries, documents and time. Then, we present the problem of temporal query performance prediction.

6.3.1 Models for Documents and Queries

We define a temporal query q as composed of two parts: keywords q_{text} and a temporal expression q_{time}. A document d consists of a textual part d_{text} (an unordered list of terms) and a temporal part d_{time} composed of the publication date and a set of temporal expression $\{t_1, \ldots t_k\}$. The publication date of d can be obtained from the function $PubTime(d)$.

Both the publication date and temporal expressions will be represented based on the time model of Berberich et al. [10] presented in Section 2.2.2.

6.3.2 Temporal Query Performance Prediction

Let q be a temporal query, D be a document collection, T be a set of all temporal expressions in D. N_D is the total number of documents in D and N_T is the number of all distinct temporal expressions in T. Temporal query performance prediction is aimed at predicting the retrieval effectiveness for q. Because q is strongly time-dependent, both the statistics of the document collection D and the set of temporal expressions T must be taken into account. Temporal query performance prediction is defined as $f(q, D, T) \rightarrow [0, 1]$, where f is a prediction function (so-called a predictor) giving a predicted score that can indicate the effectiveness of q. We are only interested in pre-retrieval predictors because they predict query performance independently from a ranking method as opposed to post-retrieval predictors.

Ultimately, we aim at finding f that can best predict the effectiveness of q, i.e., predicted scores are *highly correlated* with actual effectiveness scores. In general, f can be modeled using simple linear regression, which models the relationship between the effectiveness y with a single predictor variable p. Given N queries, simple linear regression fits a straight line through the set of N points of effectiveness scores versus predicted scores. Such that, the sum of squared residuals of the model (or vertical distances between the points of the data set and the fitted line) is as small as possible.

6.4 Pre-retrieval Predictors

In the following, we will describe existing *keyword-based* predictors proposed in previous work. The first predictor is the averaged number of characters in a query *AvQL* [89]. A key idea is that the higher the averaged length of a query, the more specific it is. For instance, World Cup soccer South Africa is more specific than World Cup.

Because of the simplicity of *AvQL*, they do not take into account the term statistics in a document collection, which can yield inaccurate prediction. The term statistics that is commonly used for measuring the specificity of query is the document frequency *df* and term frequency *tf*. The document-frequency based predictor *AvIDF* [25] determines the specificity of q_{text} by measuring the inverse document frequency *idf* for each query term and then calculating the averaged value for all query terms.

$$AvIDF = \frac{1}{|q_{text}|} \sum_{w \in q_{text}} \log \frac{N_D}{df(w)} \tag{6.1}$$

where $|q_{text}|$ is the number of query terms constituting q_{text}. N_D is the total number of documents in the collection. $df(w)$ is a document frequency of a query term w. In addition, the maximum value *MaxIDF* among all terms will also be used as a predictor. The

term-frequency based predictor measures the averaged inverse collection term frequency or *AvICTF* [46], which is computed as:

$$AvICTF = \frac{1}{|q_{text}|} \sum_{w \in q_{text}} \log \frac{N_W}{tf(w)} \tag{6.2}$$

where N_W is the total number of term occurrences in the collection. $tf(w)$ is a term frequency of a query term w in the collection. While the predictors mentioned above use either term frequencies or document frequencies, the summed collection query similarity *SumSCQ* [143] exploits both term frequencies and document frequencies by combing them together defined as follows:

$$SumSCQ = \sum_{w \in q_{text}} (1 + \ln tf(w)) \times \ln(1 + \frac{N_D}{df(w)}) \tag{6.3}$$

Intuitively, *SumSCQ* is aimed at capturing the similarity between a query and the collection by summing over all query terms. The Clarity Score is a post-retrieval method and originally proposed in [25], and the simplified (pre-retrieval) version of Clarity Score proposed in [46] can be computed as:

$$\begin{aligned} SCS &= \sum_{w \in q_{text}} P(w|q_{text}) \times \log \frac{P(w|q_{text})}{P(w)} \\ &\approx \sum_{w \in q_{text}} \frac{1}{|q_{text}|} \times \log \frac{1}{|q_{text}|} \cdot \frac{N_W}{tf(w)} \end{aligned} \tag{6.4}$$

where $P(w|q_{text})$ is the maximum likelihood of a query term w in q_{text}. Next, the sum of query weight deviation *SumVAR* is originally proposed in [143]. A term weight can be any weighting function and we use the Lucene ranking function in this chapter. *SumVAR* is given as:

$$SumVAR = \sum_{w \in q_{text}} \sqrt{\frac{1}{|D_w|} \times \sum_{d \in D_w} (tfidf(w,d) - \overline{tfidf}(w))^2} \tag{6.5}$$

where D_w is a set of documents containing a query term w, and the size of D_w, or $|D_w|$, is equal to $df(w)$. Besides, the averaged value of the equation above is also used as a predictor:

$$AvVAR = \frac{1}{|q_{text}|} \times SumVAR \tag{6.6}$$

While the predictors described previously ignore the relationship among query terms, *AvPMI* is the averaged value of pointwise mutual information (PMI), and it represents the relationship among all terms in a query. *PMI* can be computed using the statistics of term co-occurrences as follows:

$$PMI(w_i, w_j) = \log \frac{P(w_i, w_i)}{P(w_i) \cdot P(w_j)} \tag{6.7}$$

where $P(w_i, w_i)$ is the joint probability of w_i and w_j or the probability the two terms co-occur in the same document. $P(w)$ is a probability of a query term w occurring in the collection. The higher *PMI* score, the more significant the co-occurrence of two terms is than by chance. Note that, *PMI* will be computed for every query term pair (w_i, w_j), and then averaged in order to obtain *AvPMI*.

A query can have high effectiveness even if the averaged PMI is low, but there is at least one pair of query terms with a high PMI. Thus, we will also use the maximum PMI scores *MaxPMI* of all query term pairs as a predictor. The limitation of this method is that *PMI* cannot be determined for a single term query. Thus, a single term will be assigned the score 0.

6.5 Time-based Predictors

In this section, we propose 10 time-based pre-retrieval predictors: *T-AvQL*, *T-AvIDF*, *T-MaxIDF*, *T-AvICTF*, *T-SumSCQ*, *T-SCS*, *T-SumVAR*, *T-AvVAR*, *T-AvPMI* and *T-MaxPMI*. The first time-based predictor is the averaged time span of a query, or *T-AvQL*. Intuitively, the shorter the time span, the better a query will perform. *T-AvQL* is the averaged time span of all temporal expressions in q_{time} and computed as follows:

$$\textit{T-AvQL} = \frac{1}{|q_{time}|} \sum_{t \in q_{time}} \frac{(tb_l - te_l) + (tb_u - te_u)}{2} \tag{6.8}$$

where $|q_{time}|$ is the number of temporal expressions in q_{time}. For example, the query mac os x [24 march 2001] is more specific than michael jackson [1982]. As shown in Figure 6.1, the first query has smaller *T-AvQL* score and hence it performs better than the latter query.

In addition to the averaged time span of a query, the specificity of a query can be measured by the *inverse document frequency* or *idf* of a temporal expression t which indicating the general importance of t with respect to the document collection D. A time-based predictor employing *idf* denoted as *T-AvIDF* is the averaged INQUERY *idf* value over all temporal expressions in q_{time} and computed as follows:

$$\textit{T-AvIDF} = \frac{1}{|q_{time}|} \sum_{t \in q_{time}} \frac{\log(N_D + 0.5)/\textit{df}(t)}{\log(N_D + 1)} \tag{6.9}$$

where N_D is the total number of documents in D and $\textit{df}(t)$ is the number of documents containing a temporal expression t. The higher *T-AvIDF* score, the better a query should perform as illustrated in Figure 6.2.

Besides, the maximum value *T-MaxIDF* among all temporal expressions in q_{time} is also considered. Alternatively, we can determine the specificity of t by measuring the averaged *inverse collection time frequency* denoted *T-AvICTF* that is computed as:

$$\textit{T-AvICTF} = \frac{1}{|q_{time}|} \sum_{t \in q_{time}} \log \frac{N_T}{\textit{tf}(t)} \tag{6.10}$$

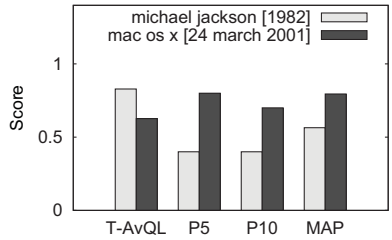


Figure 6.1: Predicted score *T-AvQL* and the retrieval effectiveness (P5, P10, MAP) of the query: mac os x [24 march 2001] vs. michael jackson [1982].

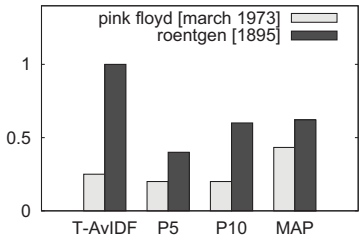


Figure 6.2: Predicted score *T-AvIDF* and the retrieval effectiveness (P5, P10, MAP) of the query: roentgen [1895] vs. pink floyd [march 1973].

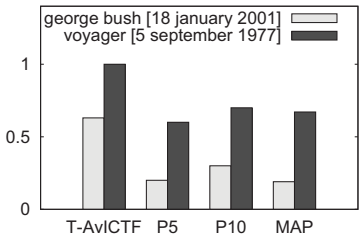


Figure 6.3: Predicted score *T-AvICTF* and the retrieval effectiveness (P5, P10, MAP) of the query: voyager [5 september 1977] vs. george bush [18 january 2001].

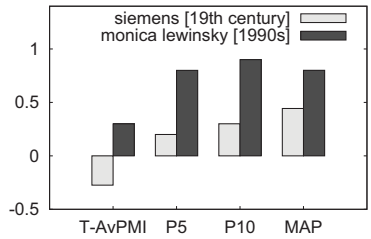


Figure 6.4: Predicted score *T-AvPMI* and the retrieval effectiveness (P5, P10, MAP) of the query: monica lewinsky [1990s] vs. siemens [19th century].

where $tf(t)$ is the total number of occurrences of t in T. Basically, *T-AvICTF* is similar to *T-AvIDF*, but *tf* is used instead of *df*. As depicted in Figure 6.3, a query with the higher score of *T-AvICTF* gains the better effectiveness.

Similar to *SumSCQ* [143], we combine both the *time frequency* and the *inverse document frequency* denoted *T-SumSCQ*, which results in the following:

$$\textit{T-SumSCQ} = \sum_{t \in q_{time}} (1 + \ln \textit{tf}(t)) \times \ln(1 + \frac{N_D}{\textit{df}(t)}) \tag{6.11}$$

In [25], *query clarity* is defined as the speciality/ambiguity of a query, and the clarity score is proposed to measure the coherence of the language usage in documents, whose models are likely to generate the query. Nevertheless, the computation of the clarity score time-consuming because it depends on the ranked list of results. Thus, the simplified version of Clarity Score is proposed in [46], where *query clarity* is only computed with respect to the query itself without the actual retrieval. We incorporate time into the simplified Clarity Score, which is denoted as *T-SCS* and computed as follows:

$$\begin{aligned} \textit{T-SCS} &= \sum_{t \in q_{time}} P(t|q_{time}) \times \log \frac{P(t|q_{time})}{P(t)} \\ &\approx \sum_{t \in q_{time}} \frac{1}{|q_{time}|} \times \log \frac{1}{|q_{time}|} \cdot \frac{N_T}{\textit{tf}(t)} \end{aligned} \tag{6.12}$$

Notice that, when q_{time} is provided with only *one* temporal expression, *T-SCS* will be actually equivalent to *T-AvICTF* and temporal Kullback-Leibler divergence (*temporalKL*) proposed by Jones and Diaz [53]. Given a query with only one temporal expression

provided, *temporalKL* can be derived as follows:

$$\begin{aligned} \textit{temporalKL} &= \sum_{t \in T} P(t|q_{\textit{time}}) \times \log \frac{P(t|q_{\textit{time}})}{P(t|T)} \\ &= \sum_{t \neq q_{\textit{time}}} P(t|q_{\textit{time}}) \times \log \frac{P(t|q_{\textit{time}})}{P(t|T)} + \sum_{t = q_{\textit{time}}} P(t|q_{\textit{time}}) \times \log \frac{P(t|q_{\textit{time}})}{P(t|T)} \\ &= 0.0 + 1.0 \times \log \frac{1.0}{P(t|T)} \\ &= \log \frac{N_T}{\textit{tf}(t)} \end{aligned} \tag{6.13}$$

$$P(t|q_{\textit{time}}) \begin{cases} 0 & \text{if } t \neq q_{\textit{time}}, \\ 1 & \text{if } t = q_{\textit{time}}. \end{cases} \tag{6.14}$$

The next time-based predictor is *T-SumVAR*, which is similar to the sum of query weight deviation [143] where *T-SumVAR* estimates how difficult it is for the retrieval model to rank documents containing query terms by examining *temporal weights* instead of *term weights* that can be performed using any temporal similarity function. We use *TSU* (cf. Chapter 4) to measure *temporal weights*. Finally, *T-SumVAR* when incorporating *temporal weights* can be computed as follows:

$$\textit{T-SumVAR} = \sum_{t \in q_{\textit{time}}} \sqrt{\frac{1}{|D_t|} \times \sum_{d \in D_t} (\textit{TSU}(t, \textit{PubTime}(d)) - \overline{\textit{TSU}}(t))^2} \tag{6.15}$$

where D_t are documents containing t and $|D_t|$ is the size of D_t, or $df(t)$. The averaged value *T-AvVAR* of the sum of query *temporal weight* deviation is also considered.

While the time-based predictors described previously ignore the relationship between query terms and time. Consider monica lewinsky [1990s] and siemens [19th century] in Figure 6.4, the first query should perform better than the latter query because the term monica lewinsky and the temporal expression *1990s* co-occur in a collection more often than by chance, while siemens and the temporal expression *19th century* rarely occur together. To determine the relationship, the pointwise mutual information (PMI) value between every query term $w \in q_{\textit{text}}$ and time $t \in q_{\textit{time}}$ can be computed as done in [42]. *T-AvPMI* is the averaged value of the PMI score of a temporal expression and all query terms, which is computed as:

$$\textit{T-AvPMI} = \frac{1}{|q_{\textit{time}}|} \sum_{t \in q_{\textit{time}}} \sum_{w \in q_{\textit{text}}} \log \frac{P(t, w)}{P(t) \cdot P(w)} \tag{6.16}$$

The maximum PMI denoted *T-MaxPMI* is also considered in a case that the averaged PMI value is low, but at least one pair of query term and time has a high PMI score.

6.6 Combination of Predictors

Using a single predictor alone might limit the predicting performance. Thus, the combination of multiple predictors can be performed as described by Jones and Diaz [53] using two different models:

1. linear regression
2. neural networks

Linear regression. A linear regression model [41] assumes that the regression function $\mathrm{E}(Y|X)$ is *linear* in the input predictors $X_1, \ldots, X_p$. Given a vector of input predictors:

$$X^T = (X_1, \ldots, X_p) \tag{6.17}$$

An output value Y (i.e., the retrieval effectiveness) will be predicted. The linear regression model is defined as:

$$f(X) = \beta_0 + \sum_{j=1}^{p} X_j \beta_j \tag{6.18}$$

where the β_j's are unknown parameters or coefficients. This model assumed *linear* relationships among predictors and aimed at predicting *linear* changes in the retrieval effectiveness.

A linear regression model and its parameters can be estimated by using a set of training data $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ of N sampled queries and each $x_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^T$ is a vector of feature scores of the i^{th} predictor. An approach to estimating the linear regression model is *least squares* that the coefficients $\beta = (\beta_0, \beta_1, \ldots, \beta_p)^T$ is chosen to minimize the residual sum of squares (RSS), or the sum of squared errors of prediction given as:

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - f(x_i))^2 \tag{6.19}$$

Intuitively, *least squares* find the best linear fit to the training data.

Neural networks. It is possible that the underlying relationships among multiple predictors are *non-linear*. In this case, multiple predictors are combined using neural networks, which model non-linear relationships from sub-combinations of predictors represented using hidden layers. A neural network is a two-stage regression or classification model [41]. For regression, there is only one output for the model, while for classification, there can be multiple output units (i.e. different classes). In our case, neural networks are applied to regression. The important units in the middle of the network (see Figure 11.2 in [41]) are the features Z_m, called *hidden units*. The values of Z_m are not directly observed, but they can be derived from transforming the original input predictors X.

Derived features Z_m are created from linear combinations of the input predictors X, and then a target output Y, i.e., the retrieval effectiveness, is modeled as a function of linear combinations of Z_m as follows.

$$\begin{aligned} Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, \ldots, M \\ Y &= \beta_0 + \beta^T Z, \\ f(X) &= g(Y) \end{aligned} \tag{6.20}$$

$Z = (Z_1, \ldots, Z_m)$. $\sigma(v)$ is the activation function that is usually chosen to be the sigmoid $\sigma(v) = 1/(1 + e^{-v})$ and sometimes Gaussian radial basic functions [41]. $g(Y)$ is the transformation function giving the final output Y. For regression, typically the identity function $g(Y) = Y$ is selected.

6.7 Evaluation

In this section, we evaluate different single predictors and the combined methods. We first will describe the setting for evaluation and then discuss the results of evaluation.

6.7.1 Setting

Document collection. We used the New York Times Annotated Corpus as a temporal document collection. Temporal expressions were extracted by annotating documents with TimeML using the TARSQI Toolkit [119]. To index documents, the Apache Lucene [5] search engine version 2.9.3 was used.

Temporal queries. We used the same dataset of queries and relevance assessments as the work by Berberich et al [10]. The set contains 40 temporal queries obtained using the Amazon Mechanical Turk. Queries were created with 5 different temporal granularities: *day*, *month*, *year*, *decade* and *century*. Due to the small number of queries for each granularity, we divided queries into 2 main categories based on their temporal granularities. The queries with *day*, *month* or *year* were grouped into the category "short period" denoted *SP*, and queries with *decade* or *century* as "long period" queries denoted *LP*. Note that, *year* was considered a short period because we found that its predicted scores were more correlated with (*day*, *month*) than (*decade*, *century*).

Retrieval model. Our goal of query prediction methods is to predict the retrieval effectiveness of a query, i.e. mean average precision (MAP) computed with respect to a particular temporal ranking model. Temporal ranking was performed using two retrieval modes as defined by Berberich et al [10]: 1) *inclusive* and 2) *exclusive*. For *inclusive*, both query terms and a temporal expression comprise the keyword part of a query q_{text}. For *exclusive*, only query terms constitutes the keyword part of a query q_{text}, and a temporal expression is excluded from q_{text}. We denote a retrieval mode as r-mode. Any temporal ranking method can be employed to retrieve documents. However, in this chapter, documents are retrieved and ranked using the method *TSU* (cf. Chapter 4) with

r-mode=*inclusive* only. Similarly, the predicted scores were computed in two modes: *inclusive* and *exclusive*. We denote a mode of prediction as p-mode.

Parameter setting. We represented the textual similarity in Eq. 4.1 using the Lucene tf-idf similarity function with default parameters. The parameters used for the temporal ranking method *TSU* were: $DecayRate = 0.5$, $\lambda = 0.5$, and $\mu = 6$ months. We used the mixture parameter $\alpha = 0.6$, which gave the best results. Most of the predictors explained in Section 6.5 were parameter-free, except the sum of query *temporal weight* deviation *T-SumVAR* that employs the temporal ranking *TSU*. The Weka implementation [131] was used for modeling simple linear regression, linear regression and neural networks, and we used the recommended values [53] for the parameters. For linear regression, the selection method was M5 and the ridge parameter was a default (1.0e-8). For neural networks, a learning rate was 0.3 and a momentum rate for the back propagation algorithm was 0.2. The network was trained using 500 training epochs. All prediction models were trained using cross-validation of 5 folds with 10 repetitions.

Metrics. Correlation coefficient and root mean squared error (RMSE) were measured as the evaluation of predicting performance. Correlation coefficient [131] measures the statistical correlation between the predicted scores and the MAP scores, which ranges from 1 for perfectly correlated results, through 0 when there is no correlation, to -1 when the results are perfectly correlated negatively. RMSE indicates how far the predicted scores deviate on average from the MAP scores, as done in the previous work [42, 53]. Given $\hat{Y}$ be the predicted scores and Y be the MAP scores of queries:

$$RMSE = \sqrt{\frac{1}{p}\sum_{i=1}^{p}(y_i - \hat{y}_i)^2} \tag{6.21}$$

The lower the RMSE value, the better a predictor.

6.7.2 Results

The performance of single predictors are shown in Table 6.1. For each predictor, the results are reported for two prediction modes, i.e., when p-mode=*inclusive* and p-mode=*exclusive*. For each p-mode, the results for "short period" and "long period" are displayed. The performance of each predictor is statistically tested with the worst performed predictor using paired t-test with significant at $p < 0.05$. Because each query in the dataset is provided with only one temporal expression, we omit the results for some predictors. For example, we do not show the results of *T-MaxIDF* because they are exactly the same as those of *T-AvIDF*. Note that, a negative value of correlation coefficient does not always imply that a predictor performs worst, but it means that the predictor is highly correlated negatively (closer to -1) when the absolute value is sufficiently high.

The results in Table 6.1 show that the correlation coefficient of keyword-based predictors differs between two prediction modes. This is because the predicted scores of keyword-based predictors with p-mode=*inclusive* are different from those with p-mode=*exclusive*, while only the results of two time-based predictors (*T-AvPMI* and *T-MaxPMI*) are changed

with a prediction mode. Generally, predictors perform better with p-mode=*exclusive*, that is, when a temporal expression is excluded from q_{text}.

In addition, predicting performance varies according to time granularities. For "short period" in both p-modes, the best performing keyword-based predictors are *AvQL*, *MaxPMI* and *AvPMI*, while the best performing timed-based predictors are *T-AvICTF*, *T-AvIDF* and *T-AvPMI*. For "long period", among keyword-based predictors, *SumSCQ* is the best performing predictor with p-mode=*exclusive*, while *MaxIDF* is the best performing predictor with p-mode=*inclusive*. In addition, *T-SumSCQ* outperforms all other predictors significantly for "long period" in both modes. Generally, most of predictors perform slightly better with p-mode=*exclusive*. For "short period", the predicting performance of the keyword-based predictor *AvQL* and the time-based predictor *T-AvICTF* is quite similar, whereas *SumSCQ* and *T-SumSCQ* perform quite differently for "long period".

Table 6.1 shows the predicting performance of single predictors measured in RMSE. The results indicate that the high values of correlation coefficient do not always imply the low values of RMSE. Similar to the results of correlation coefficient, *AvQL* and *T-AvICTF* are among the best performing predictors for "short period". Notice that, *T-AvIDF* is the worst predictor for "long period" and its RMSE value is too high (=0.65). We performed an error analysis and found that the predicted scores of *T-AvIDF* of queries in the class "long period" are very small, which yields high difference between the predicted scores and the actual MAP scores.

Table 6.2 shows the performance of the combined methods using linear regression† and neural networks∓. The results of the combined predictors are statistically tested with those of the best performing single predictors, i.e., (*AvQL*,*T-SumSCQ*) and (*T-ICTF*,*T-MaxPMI*) as measured by correlation coefficient and RMSE respectively. In Table 6.2, the predicting performance of the best single predictors are given in parentheses. Each time-based predictor is combined with its corresponding keyword-based predictor. For instance, *T-AvQL*† denotes the combination of *T-AvQL* and *AvQL* using linear regression. The combination of all predictors is denoted *ALL*. The results of combined methods with respect to correlation coefficient are as follows. For "short period" and p-mode=*inclusive*, the methods *T-AvQL*† *T-AvQL*∓, *ALL*† and *T-MaxPMI*† outperform the best single predictor significantly. For "short period" and p-mode=*exclusive*, only *T-AvQL*∓ performs significantly better than the best single method.

In general, for "long period' we do not gain any improvement for the combined methods since the correlation coefficient of the best performing predictor *T-SCQ* is relatively high (though it is negative). Notice that, combining *T-SumSCQ* with *SumSCQ* results in worse correlation coefficient (=-0.37) when trained using neural network. Consider the performance measured using RMSE. For "short period", the combined methods *T-AvQL*† and *T-MaxPMI*† are significantly better than *AvQL*. For "long period", all combined methods do not gain better performance compared to that of *T-MaxPMI*. We conclude our findings as follows.

1. Predictors perform better with p-mode=*exclusive*, that is, when temporal expression are excluded from q_{text}.

2. Time-based single predictors are good for predicting the performance of "short pe-

Table 6.1: Performance of single predictors measured using correlation coefficient and root mean squared error (RMSE); in bold indicates statistically different from the baseline predictor (as underlined) using t-test with significant at $p < 0.05$.

Predictor	Correlation coefficient				RMSE			
	inclusive		*exclusive*		*inclusive*		*exclusive*	
	SP	*LP*	*SP*	*LP*	*SP*	*LP*	*SP*	*LP*
AvQL [89]	**0.36**	0.27	**0.39**	-0.02	**0.28**	**0.23**	**0.29**	**0.25**
AvIDF [25]	**-0.26**	0.04	-0.20	0.12	0.30	**0.24**	**0.29**	**0.24**
MaxIDF [46]	0.04	**-0.27**	-0.16	**-0.27**	0.29	**0.25**	0.30	**0.25**
AvICTF [46]	-0.13	0.19	-0.18	0.24	0.30	**0.22**	**0.29**	**0.23**
SCS [46]	-0.14	0.21	-0.14	0.24	0.30	**0.22**	**0.29**	**0.23**
SumSCQ [143]	-0.09	-0.05	**0.16**	**-0.45**	0.29	**0.24**	**0.29**	**0.24**
SumVAR [143]	-0.20	0.07	**-0.31**	0.19	0.30	**0.22**	0.31	**0.22**
AvVAR [143]	**-0.20**	0.23	**-0.35**	0.00	0.30	**0.23**	**0.30**	**0.23**
AvPMI [42]	**0.29**	-0.05	**0.28**	0.02	0.30	**0.24**	**0.28**	**0.24**
MaxPMI [42]	**0.32**	-0.06	**0.35**	-0.04	0.28	**0.24**	**0.28**	**0.24**
T-AvQL	**0.19**	0.05	**0.19**	0.05	**0.28**	**0.24**	**0.28**	**0.24**
T-AvIDF	**0.27**	-0.05	**0.27**	-0.05	0.29	0.65	**0.29**	0.65
T-AvICTF	**0.35**	0.08	**0.35**	0.08	**0.27**	**0.25**	**0.27**	**0.25**
T-SumSCQ	-0.02	**-0.59**	-0.02	**-0.59**	0.29	**0.32**	**0.29**	**0.32**
T-SumVAR	**0.21**	-0.07	**0.21**	-0.07	0.28	**0.24**	**0.28**	**0.24**
T-AvPMI	**0.15**	0.23	**0.28**	0.20	0.30	**0.22**	**0.27**	**0.23**
T-MaxPMI	0.02	0.08	0.13	0.08	**0.29**	**0.21**	0.27	**0.21**

riod" queries.

3. Some combination methods can improve the predicting performance. However, it is not clear if our proposed time-based predictors and the combination methods are good for predicting the performance of "long period" queries.

6.8 Conclusions

In this chapter, we have studied the problem of predicting query performance for temporal search. We have exploited both textual and temporal information to predict query performance more accurately, and proposed time-based predictors as analogous to keyword-based predictors. In order to improve the predicting performance of single predictors, we have employed two approaches for combining multiple predictors; linear regression and neural networks. The proposed predictors and the combination methods have been evaluated, and the experimental results showed that our time-based predictors are among the best performing single predictors. In addition, the combination methods significantly improve the performance of the best single predictors.

Table 6.2: Performance of combined predictors measured using correlation coefficient and root mean squared error (RMSE); in bold indicates statistically different from the performance of best single predictors (as provided in parentheses) using t-test with significant at $p < 0.05$.

Predictor	Correlation coefficient				RMSE			
	inclusive		*exclusive*		*inclusive*		*exclusive*	
	SP (0.36)	*LP* (-0.59)	*SP* (0.39)	*LP* (-0.59)	*SP* (0.27)	*LP* (0.21)	*SP* (0.27)	*LP* (0.21)
T-AvQL†	**0.50**	-0.07	0.33	-0.10	**0.26**	0.25	0.28	0.24
T-AvIDF†	-0.04	0.01	-0.06	0.02	0.29	0.23	0.29	0.23
T-MaxIDF†	-0.10	0.01	-0.05	0.01	0.30	0.23	0.30	0.23
T-AvICTF†	-0.02	-0.22	-0.02	-0.19	0.30	0.26	0.29	0.26
T-SCS†	-0.02	-0.16	-0.02	-0.19	0.29	0.25	0.29	0.26
T-SumSCQ†	-0.04	-0.16	-0.04	-0.19	0.29	0.25	0.29	0.32
T-SumVAR†	-0.07	-0.08	-0.07	-0.08	0.29	0.23	0.29	0.23
T-AvVAR†	-0.06	-0.07	-0.04	-0.07	0.30	0.23	0.29	0.23
T-AvPMI†	-0.10	0.03	0.41	0.03	0.33	0.24	**0.27**	0.23
T-MaxPMI†	**0.36**	-0.05	0.30	-0.10	0.26	0.23	0.28	0.23
ALL†	**0.43**	-0.04	0.29	-0.11	0.34	0.32	0.33	0.26
T-AvQL∓	**0.47**	0.13	**0.50**	-0.06	0.30	0.27	0.30	0.26
T-AvIDF∓	0.10	-0.32	0.04	-0.26	0.41	0.29	0.34	0.34
T-MaxIDF∓	-0.02	-0.29	-0.05	-0.29	0.36	0.27	0.37	0.27
T-AvICTF∓	0.12	-0.17	0.22	0.01	0.33	0.26	0.30	0.29
T-SCS∓	0.13	-0.09	0.24	-0.07	0.33	0.26	0.31	0.30
T-SumSCQ∓	-0.06	-0.09	-0.11	-0.37	0.33	0.26	0.34	0.24
T-SumVAR∓	-0.09	-0.03	-0.14	0.03	0.35	0.23	0.37	0.24
T-AvVAR∓	-0.06	-0.05	-0.02	-0.10	0.34	0.23	0.35	0.24
T-AvPMI∓	0.11	0.16	0.41	0.16	0.36	0.27	0.30	0.25
T-MaxPMI∓	0.32	0.18	0.50	0.23	0.31	0.23	0.29	0.23
ALL∓	0.22	-0.09	0.17	0.00	0.38	0.45	0.44	0.42

Chapter 7

Time-aware Ranking Prediction

Two time dimensions commonly exploited in time-aware ranking are 1) *publication time*, and 2) *content time* (temporal expressions mentioned in documents' contents). As shown in the chapter, it makes a difference in retrieval effectiveness for temporal queries when ranking using publication time or content time. By determining whether a temporal query is sensitive to publication time or content time, the most suitable retrieval model can be employed. In this chapter, we address the research question: *how to predict the suitable time-aware ranking model for a temporal query?*

7.1 Motivation

Several studies of real-world user query logs have shown that temporal queries comprises a significant fraction of web search queries [87, 95, 141]. For example, Zhang et al. [141] showed that 13.8% of queries contain explicit time and 17.1% of queries have temporal intent implicitly provided. An example of a query with time explicitly provided is U.S. Presidential election 2008, while Germany FIFA World Cup is a query without temporal criteria provided. However, for the latter example, a user's temporal intent is *implicitly* provided, i.e., referring to the world cup event in 2006. As has been shown in previous work [10], incorporating the time dimension into the ranking models can significantly improve query effectiveness in the case of temporal queries.

Two time dimensions that are commonly exploited in time-aware ranking are 1) *publication time*, and 2) *content time* (temporal expressions mentioned in documents' contents). We denote a time-aware ranking model that exploits publication time as *PT-Rank*, and a time-aware ranking model that exploits content time as *CT-Rank*. As we will show in more detail later in this chapter, which ranking model to use has high impact on retrieval effectiveness for temporal queries. As an example, consider the four queries in Table 7.1. We find that the queries iraq 2001 and mac os x 24 march 2001 perform better using *PT-Rank*, while the queries sound of music 1960s and michael jackson 1982 perform best using *CT-Rank*. Therefore, it is important to determine whether a temporal query is sensitive to publication time, or content time, so that we can choose the suitable ranking

Table 7.1: Retrieval effectiveness of queries with respect to *PT-Rank* and *CT-Rank*.

Query	MAP	
	PT-Rank	*CT-Rank*
iraq 2001	**0.60**	0.40
sound of music 1960s	0.11	**0.29**
mac os x 24 march 2001	**0.79**	0.36
michael jackson 1982	0.56	**0.65**

model that gives the best results for the query.

In this chapter, we propose an approach to predicting the suitable time-aware ranking model using machine learning techniques. We learn a prediction model using three classes of features obtained from analyzing top-k retrieved documents, i.e., an analysis of documents' contents, time distribution and retrieval scores.

Contributions

Our main contributions in this chapter are:

- We perform the first study on the impact on retrieval effectiveness of ranking models using the two time dimensions for temporal queries.
- We propose an approach to predicting the suitable time-aware ranking model based on machine learning techniques, using three classes of features.

Organization

The organization of the rest of the chapter is as follows. In Section 7.2, we give an overview of related work. In Section 7.3, we discuss classification of queries, documents and query models, and present the time-aware ranking models used in the subsequent parts of the chapter. In Section 7.4, we present our proposed features used to learn a ranking prediction model. In Section 7.5, we evaluate our approach to predicting a ranking model by conducting extensive experiments. Finally, in Section 7.6, we conclude the chapter.

7.2 Related Work

A number of ranking models exploiting temporal information have been proposed, including [7, 10, 33, 54, 74, 87]. In [74], Li and Croft incorporated time into language models, called time-based language models, by assigning a document prior using an exponential decay function of the publication time of document, i.e., the creation date. They did not have temporal information needs explicitly provided, but they focused on recency queries. The work by Baeza-Yates [7] proposed to extract temporal expressions from news, index news articles together with temporal expressions, and retrieve temporal information (in this case, future-related events) by using a probabilistic model. A document score is

given by multiplying a *keyword* similarity and a time confidence, i.e., a probability that the document's events will actually happen.

In [54], Kalczynski and Chou proposed a temporal retrieval model for news archives. In their work, temporal expressions in a query and documents were explicitly modeled in ranking. A query is defined as a set of precise temporal information needs, i.e., the finest time chronon, or a day. Thus, they assumed that the uncertainty applied only to temporal references in documents, and it was represented as a fuzzy set function. Metzler et al. [87] considered implicit temporal information needs. They proposed mining query logs and analyze query frequencies over time in order to identify strongly time-related queries. They presented a ranking model concerning implicit temporal needs, and the experimental results showed the improvement of the retrieval effectiveness of temporal queries for web search.

Berberich et al. [10] integrated temporal expressions into query-likelihood language modeling, which considers uncertainty inherent to temporal expressions in a query and documents. That is, temporal expressions can refer to the same time interval even they are not exactly equal. The work by Berberich et al. required explicit temporal information needs as a part of query. The most relevant work for us is the work by Jones and Diaz [53]. They proposed features for classifying queries into three temporal classes, i.e., atemporal, temporally unambiguous (recency or historic) and temporally ambiguous (periodic). They analyzed document collections and proposed four features for classifying queries: temporal KL-divergence, autocorrelation, statistics of the rank order and burst model. As opposed to our work, we want to classify a temporal query based the two time dimensions, rather than using *temporal patterns* as done by Jones and Diaz [53].

7.3 Preliminaries

In the following, we present a classification of queries based on the two time dimensions. Finally, we describe the models for documents and queries, and outline the time-aware ranking models used in this chapter.

7.3.1 Classification of Queries

A query can be categorized into two main classes: *temporal* and *non-temporal*. Temporal queries are those that relevant documents are strongly dependent on time, e.g., World Series 2004 and NFL Draft. On the contrary, non-temporal queries are those that relevant documents are not dependent on time, e.g., muffin recipes and Hawaiian dance. Then, we further classify temporal queries into the two subclasses: publication-time sensitive (denoted *PT-sensitive*) and content-time sensitive (denoted *CT-sensitive*). Our intuition is to leverage the two most useful time dimensions for relevance ranking. *PT-sensitive* queries are those sensitive to the publication time of documents, and *CT-sensitive* queries are those sensitive to temporal expressions in documents' contents.

For example, given the query Japan quake 869 AD, relevant documents are possibly documents that contains the temporal expression 869 AD, not those dated to 869

AD. When searching for a current event (breaking news, or popular topics), temporal expressions might not be necessary because the publication time of documents are highly correlated with the event. Note that, whether to consider publication time or content time is also dependent on the time span of a document collection. For instance, when the New York Times Annotated Corpus [96] with the time span (1987-2007) is used, a query that its time is not overlapped with the collection time span must be ranked with respect to temporal expressions instead of publication time.

7.3.2 Models for Documents, Queries, and Ranking

A document d consists of a textual part d_{text} (an unordered list of terms) and a temporal part d_{time} composed of the publication date and a set of temporal expression $\{t_1, \dots t_k\}$. The publication date of d can be obtained from the function *PubTime*(d). Temporal expressions mentioned in the contents of d can be obtained from the function *ContentTime*(d). A temporal query q is composed of two parts: keywords q_{text} and a temporal expression q_{time}.

The ranking model used for *PT-Rank* and *CT-Rank* is based on a mixture model, which linearly combines textual similarity and temporal similarity for all ranking methods. Given a temporal query q, a document d will be ranked according to a score computed as follows:

$$S(q,d) = (1-\alpha) \cdot S'(q_{text}, d_{text}) + \alpha \cdot S''(q_{time}, d_{time}) \tag{7.1}$$

where the mixture parameter α indicates the importance of textual similarity $S'(q_{text}, d_{text})$ and temporal similarity $S''(q_{time}, d_{time})$. Both similarity scores must be normalized, e.g., divided by the maximum scores, in order to the final score $S(q,d)$. $S'(q_{text}, d_{text})$ can be measured using any of existing text-based weighting functions. $S''(q_{time}, d_{time})$ measure temporal similarity by assuming that a temporal expression $t_q \in q_{time}$ is generated independently from each other, and a two-step generative model was used [10]:

$$\begin{aligned} S''(q_{time}, d_{time}) &= \prod_{t_q \in q_{time}} P(t_q|d_{time}) \\ &= \prod_{t_q \in q_{time}} \left(\frac{1}{|d_{time}|} \sum_{t_d \in d_{time}} P(t_q|t_d) \right) \end{aligned} \tag{7.2}$$

Linear interpolation smoothing will be applied to give the probability $P(t_q|t_d)$ for an unseen query temporal expression t_q in d. The probability $P(t_q|t_d)$ will be computed differently for two time-aware ranking methods, i.e., *CT-Rank* and *PT-Rank*.

In this chapter, we use the LmtU ranking function [10] for computing $P(t_q|t_d)$ for *CT-Rank*. LmtU is a time-aware language modeling approach, which considers the content time of documents and time uncertainty. For *PT-Rank*, we employ the TSU ranking function (cf. Chapter 4) to compute $P(t_q|t_d)$. TSU is based on the publication time of documents and it is computed using an exponential decay function to capture time uncertainty.

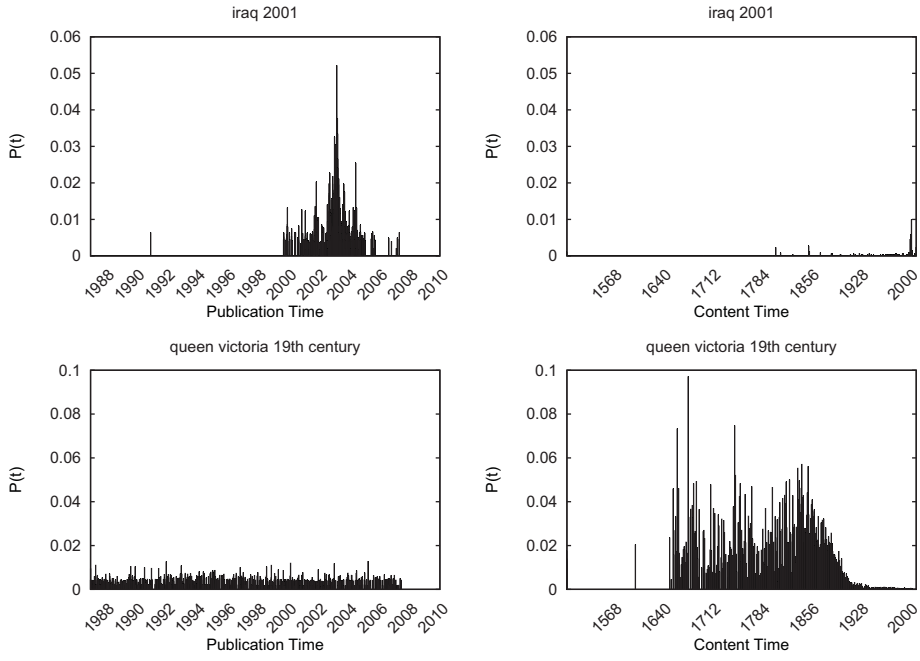


Figure 7.1: Distribution over two time dimensions of top-1000 documents for the queries iraq 2001 and queen victoria 19th century.

7.4 Ranking Prediction

As mentioned earlier, it does make a difference in retrieval effectiveness when ranking using two time dimension (as we will show through experiments in more detail). Given a temporal query, we want to predict the right time-aware ranking model that gives the best results for the query. We use different machine learning techniques to learn a ranking prediction model. In particular, we propose three classes of features obtained from analyzing top-k retrieved documents, i.e., an analysis of time distribution, documents' contents, and retrieval scores.

7.4.1 Temporal KL-divergence

Temporal KL-divergence [53] was proposed to determine temporal classes of queries, which is measured as the difference between the distribution over time of top-k retrieved documents of q and the document collection. We extend this method by using the *content time* of top-k documents for measuring temporal KL-divergence, in addition to just considering publication time as done by Jones and Diaz [53]. As shown in Figure 7.1, we observe a difference of the distribution of top-k retrieved documents over two time dimensions. Thus, it is necessary to consider both time dimensions.

We denote to the temporal KL-divergence of the publication time as KL_{PT} and the temporal KL-divergence of the content time as KL_{CT}. KL_{PT} can be computed as:

$$\begin{aligned} KL_{PT}(D_q||C,q) &= \sum_{t \in T_P} P(t|q) \cdot \log \frac{P(t|q)}{P(t|T_P)} \\ P(t|q) &= \sum_{d \in D_q} P(t|d) \cdot \frac{P(q|d)}{\sum_{d' \in D_q} P(q|d')} \end{aligned} \tag{7.3}$$

where T_P is the set of all publication dates in the document collection. Note that we use a *1-day* granularity for each publication date. $P(t|T_P)$ is the probability of a publication date t in the collection. $P(t|q)$ is the probability of generating a publication date t given q and D_q is top-k documents retrieved with respect to q. $P(t|q)$ is defined using *relevance language modeling* [73], that is, the top-k retrieved documents D_q are considered and weighed according to the document's probability of relevance, i.e, $P(q|d)$. In other word, $P(q|d)$ is a retrieval score of d for a particular ranking model.

$$P(t|d) = \begin{cases} 0 & \text{if } \textit{PubTime}(d) \neq t, \\ 1 & \text{if } \textit{PubTime}(d) = t \end{cases} \tag{7.4}$$

The temporal KL-divergence of the content time of top-k retrieved documents KL_{CT} can be computed as follows.

$$\begin{aligned} KL_{CT}(D_q||C,q) &= \sum_{t \in T_C} P(t|q) \cdot \log \frac{P(t|q)}{P(t|T_C)} \\ P(t|q) &= \sum_{d \in D_q} P(t|d) \cdot \frac{P(q|d)}{\sum_{d' \in D_q} P(q|d')} \end{aligned} \tag{7.5}$$

where T_C is a set of all temporal expressions in the document collection. $P(t|T_C)$ is the probability of a temporal expression t in the collection. $P(t|d)$ of KL_{CT} is computed differently from that of KL_{PT} because a document d can contain with more than one temporal expression. So, the probability $P(t|d)$ of KL_{CT} can be computed as:

$$P(t|d) = \frac{c(t,d)}{\sum_{t' \in d} c(t',d)} \tag{7.6}$$

where $c(t,d)$ is the number of occurrence of temporal expression t in a document d, and $\sum_{t' \in d} c(t',d)$ is the total number of occurrence of all temporal expressions in d. For both KL_{PT} and KL_{CT}, a smoothing technique [139] will be applied to $P(t|q)$ in order to avoid a problem of zero-probability.

7.4.2 Content Clarity

It has been suggested in [53] that temporal features alone could not achieve high accuracy for query classification. Thus, we also employ a feature based on an analysis of the

contents of top-k retrieved documents, such as, the content clarity [25]. A query's content clarity, which is widely used in a query performance prediction task [20]. Intuitively, the content clarity can be used for determining the specificity of a query. Generally, the more specific a query, the better it will perform. The content clarity is measured by the Kullback-Leibler (KL) divergence between the distribution of terms of retrieved documents and the background collection. More precisely, the content clarity is the KL-divergence between the query language model and the collection language model and it can be computed as follows:

$$\textit{Clarity} = \sum_{w \in V} P(w|q) \cdot \log \frac{P(w|q)}{P(w|C)} \tag{7.7}$$

where w is a term in a vocabulary V, i.e., the set of all distinct terms in the collection, $P(w|q)$ is the probability of generating w given q and $P(w|C)$ is the probability of w in the document collection. More detail of the calculation of the content clarity can be referred to [25]. The higher clarity score indicates that the query is less ambiguous and the better it will perform.

7.4.3 Retrieval Scores

Features presented above are based on an analysis of time distribution, and contents of top-k retrieved documents. An alternative method for predicting a ranking model is to analyze the retrieval scores of top-k retrieved documents [99]. The idea is to measure the divergence of retrieval scores from the base ranking, e.g., a non time-aware ranking model, is to determine the extent that a ranking model alters the scores of the initial ranking. In this chapter, we employ the Jensen-Shannon divergence (JS) for measuring the divergence of scores obtained from two ranking models. In particular, our features based on the analysis of retrieval scores are composed of: 1) averaged scores of the base ranking, 2) averaged scores of *PT-Rank*, 3) averaged scores of *CT-Rank* and 4) divergence from the base ranking model. The base ranking model is the ranking function used in the initial retrieval.

$$\textit{AVG} = \frac{1}{k} \sum_{d=1}^{k} S(q, d_i) \tag{7.8}$$

The Jensen-Shannon divergence (JS) of scores obtained from two ranking models can be computed as.

$$\begin{aligned} \textit{JS}(S_b||S_{t_i}, q) &= \frac{1}{2} \cdot \textit{KL}(S_b||S_{t_i}, q) + \frac{1}{2} \cdot \textit{KL}(S_{t_i}||S_b, q) \\ &= \sum_{d=1}^{k} S_b(d_i, q) \cdot \log \frac{S_b(d_i, q)}{\frac{1}{2} \cdot S_b(d_i, q) + \frac{1}{2} \cdot S_{t_i}(d_i, q)} \end{aligned} \tag{7.9}$$

where $S_b(d_i, q)$ is the retrieval score of a document d when ranked using the base ranking S_b, and $S_{t_i}(d_i, q)$ is the retrieval score of a document d when ranked using a candidate time-aware ranking S_{t_i}, i.e., *PT-Rank* or *CT-Rank*.

All of features explained above correspond to the two time-aware ranking models. Thus, there are five different features in total for ranking model prediction, i.e., AVG_{base}, $AVG_{PT\text{-}Rank}$, $AVG_{CT\text{-}Rank}$, $JS_{PT\text{-}Rank}$ and $JS_{CT\text{-}Rank}$.

7.5 Evaluation

In this section, we evaluate our proposed approach by conducting two experiments. We first evaluate the ranking prediction model as a classification task, and then we show how the ranking prediction help improving the retrieval effectiveness. In the following, we describe the setting of experimental evaluation, as well as explain the results of experiments.

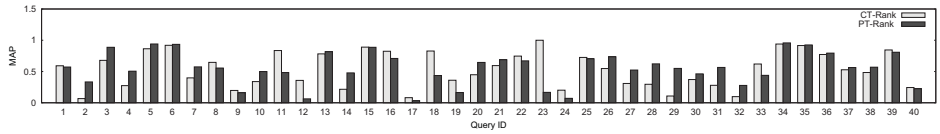
7.5.1 Setting

Document collection. We used the New York Times Annotated Corpus as a temporal document collection. Temporal expressions were extracted by annotating documents with TimeML using the TARSQI Toolkit [119] search engine version 2.9.3 was used. We used 40 temporal queries and relevance assessments [10] obtained using crowdsourcing.

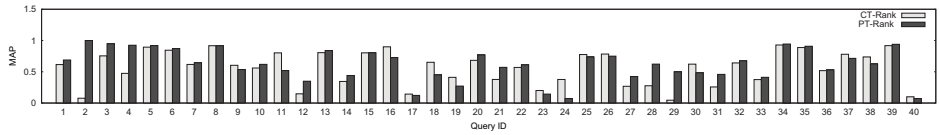
Retrieval models. The retrieval of explicit temporal queries were performed using two retrieval modes as defined in [10]: 1) *inclusive* and 2) *exclusive*. For *inclusive*, both query terms and a temporal expression comprise the keyword part of a query q_{text}. For *exclusive*, only query terms constitutes the keyword part of a query q_{text}, and a temporal expression is excluded from q_{text}. For the classification experiment, queries were labeled with two classes, i.e., *PT-sensitive* or *CT-sensitive*, by assuming the ranking model that gives the best MAP score as a query label as shown in Fig. 7.2. Note that, we excluded the queries that have a small difference of MAP for the two ranking models.

Parameter settings. We used the Lucene tf-idf similarity function [5] for computing the textual similarity in Eq. 7.1. The parameters used for the TSU ranking function were: $DecayRate = 0.5$, $\lambda = 0.5$, and $\mu = 6$ months. For the LMTU ranking function, we used the recommended value 0.75 for the smoothing parameter γ [10]. The mixture parameter α in Eq. 7.1 were empirically determined by studying the sensitivity of α and MAP as shown in Fig. 7.3. For *inclusive* mode, we selected the best performing values: $\alpha = 0.5$ for *PT-Rank* and $\alpha = 0.6$ for *CT-Rank*, while for *exclusive* mode, we used $\alpha = 0.5$ for *PT-Rank* and $\alpha = 0.1$ for *CT-Rank*.

The smoothing parameter λ for temporal KL divergence was set to 0.1. The Weka implementation [131] was used for training query classifiers and models for ranking prediction. We experimented with several classification algorithms: decision tree (J48), Naive Bayes (NB), neural network (NN) and SVM. We used the default values for the parameters of classifiers. Both classification and retrieval experiments were trained using cross-validation of 10 folds with 10 repetitions. A majority classifier was a baseline for the classification experiment.



(a) *inclusive*



(b) *exclusive*

Figure 7.2: Queries are labeled using the ranking model that gives the best MAP score when retrieved using (a) *inclusive* mode and (a) *exclusive* mode.

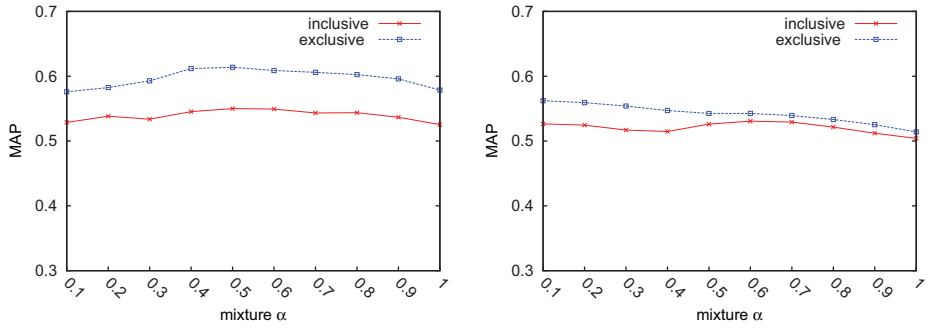
Metrics. We reported the performance of classification using accuracy. The retrieval experiment was reported using precision at 1, 5 and 10 (P@1, P@5, and P@10 respectively), and Mean Average Precision (MAP). Note that, we computed the average performance over 10 folds to measure the overall performance, for both classification and retrieval experiments.

7.5.2 Results

Classification results. We performed query classification by using several classification algorithms: decision tree (J48), Naive Bayes (NB), neural network (NN) and SVM. The results of query classification shown in Table 7.2 are conducted with respect to two retrieval modes (*exclusive* and *inclusive*). For each retrieval mode, we varied the number of top-k retrieved documents in order to study how a k-value affect the classification performance. For each case, the performance of single features and a combination of different features are shown. The baseline method for query classification is the majority classifier. The accuracy of the baseline is 0.54 for *exclusive* and 0.60 for *inclusive*.

For *exclusive*, the results of using a small number of top-k documents are in general better than a large number of top-k documents. For top-100, $JS_{PT\text{-}Rank}$ modeled using SVM outperforms the baseline classifier and all other features significantly (accuracy=0.72). For top-500, all single features perform worse compared to the baseline classifier. However, the combination of features (*ALL*, *Clarity*+KL_{PT}+KL_{CT}, and *Clarity*+$JS_{PT\text{-}Rank}$+$JS_{CT\text{-}Rank}$) shows the best performance among other methods, and improve the baseline classifier significantly (accuracy=0.65). Unfortunately, the classification results when using k=1000 are not good compared to the baseline classifier.

For *inclusive*, the results are similar to those of *exclusive*, that is, the performance of top-100 is the best among the other k's values. For top-100, the best performing feature is the combination of *Clarity*, $JS_{PT\text{-}Rank}$ and $JS_{CT\text{-}Rank}$, which gains the accuracy of 0.75. $JS_{PT\text{-}Rank}$ outperforms the baseline classifier significantly when modeled using J48, NN



(a) *PT-Rank* (b) *CT-Rank*

Figure 7.3: Sensitivity of α and MAP for *PT-Rank* and *CT-Rank*.

and SVM with the accuracy of 0.74, 0.74 and 0.68 respectively. For top-500, $JS_{PT\text{-}Rank}$ performs best among others with the accuracy of 0.71. For top-100, no method performs significantly better than the baseline classifier.

To conclude, it is obvious that using a small number of top-k documents achieves better performance than other k's values. An explanation can be that the larger number of top-k documents, the more irrelevant documents are introduced into the analysis. The performance among different feature classes shows that the retrieval-score feature, i.e., $JS_{PT\text{-}Rank}$, performs well in most case, while other feature classes based on Clarity and time distribution do not perform very well. Thus, our plans for future work include a method for selecting only *important* documents from top-k retrieved documents, and conducting a score analysis of *CT-Rank* and other temporal features.

Retrieval results. As described earlier, we aim at improving the retrieval effectiveness by predicting the suitable ranking model. For each query, we perform retrieval using the predicted ranking model. More precisely, the ranking prediction models are: 1) $JS_{PT\text{-}Rank}$ (modeled using top-100 and SVM) for retrieval in *exclusive*, and 2) *Clarity*+$JS_{PT\text{-}Rank}$+$JS_{CT\text{-}Rank}$ (modeled using top-100 and J48) for retrieval in *inclusive*. The retrieval results shown in Table 7.3 are the effectiveness of different ranking models. *Rank-Prediction* is the ranking model based on the ranking prediction models. *MAX* is the maximum (or optimal) effectiveness that can be achieved, that is, when a prediction model performs accurately 100%. The retrieval results are compared with the baseline method, i.e., *CT-Rank*. The results show that our prediction-based ranking model (*Rank-Prediction*) outperforms the baseline significantly in P@1 and MAP. However, we note that it is difficult for *Rank-Prediction* to achieve the optimal effectiveness because of the prediction accuracy as explained previously.

Table 7.2: Results of classification measured using accuracy; † indicates statistically different from the baseline majority classifier using t-test with significant at $p < 0.05$.

Top-k	Feature	*exclusive*				*inclusive*			
		J48	*NB*	*NN*	*SVM*	*J48*	*NB*	*NN*	*SVM*
100	*Clarity*	0.53	0.34	0.44	0.51	0.59	0.45	0.57	0.60
	KL_{PT}	0.53	0.40	0.41	0.53	0.60	0.59	0.47	0.60
	KL_{CT}	0.53	0.49	0.51	0.53	0.60	0.54	0.54	0.60
	AVG_{Base}	0.40	0.55	0.61	0.53	0.63	0.58	0.61	0.60
	$AVG_{PT\text{-}Rank}$	0.53	0.47	0.43	0.53	0.60	0.58	0.54	0.60
	$AVG_{CT\text{-}Rank}$	0.48	0.50	0.64	0.53	0.60	0.50	0.49	0.60
	$JS_{PT\text{-}Rank}$	0.50	0.60	0.57	**0.72**†	**0.74**†	0.64	**0.74**†	**0.68**†
	$JS_{CT\text{-}Rank}$	0.55	0.45	0.45	0.38	0.60	0.58	0.55	0.54
	$Clarity+KL_{PT}+KL_{CT}$	0.54	0.38	0.46	0.54	0.61	0.36	0.43	0.61
	$Clarity+JS_{PT\text{-}Rank}+JS_{CT\text{-}Rank}$	0.50	0.38	0.46	0.42	**0.75**†	0.46	0.61	0.64
	ALL	0.42	0.36	0.30	0.50	0.58	0.40	0.37	0.63
500	*Clarity*	0.51	0.36	0.44	0.53	0.60	0.40	0.59	0.60
	KL_{PT}	0.53	0.44	0.46	0.53	0.59	0.58	0.49	0.60
	KL_{CT}	0.53	0.49	0.44	0.53	0.60	0.53	0.59	0.60
	AVG_{Base}	0.53	0.53	0.49	0.53	0.56	0.59	0.62	0.60
	$AVG_{PT\text{-}Rank}$	0.53	0.59	0.47	0.53	0.60	0.55	0.55	0.60
	$AVG_{CT\text{-}Rank}$	0.53	0.50	0.48	0.53	0.59	0.50	0.49	0.60
	$JS_{PT\text{-}Rank}$	0.50	0.50	0.51	0.42	0.64	0.66	**0.71**†	0.57
	$JS_{CT\text{-}Rank}$	0.53	0.44	0.44	0.42	0.60	0.56	0.54	0.57
	$Clarity+KL_{PT}+KL_{CT}$	0.63	0.55	0.63	**0.65**†	0.61	0.46	0.46	0.61
	$Clarity+JS_{PT\text{-}Rank}+JS_{CT\text{-}Rank}$	**0.65**†	0.55	0.60	**0.65**†	0.61	0.46	0.43	0.61
	ALL	0.64	0.59	0.47	0.**65**†	0.52	0.45	0.41	0.60
1000	*Clarity*	0.53	0.36	0.46	0.53	0.60	0.42	0.60	0.60
	KL_{PT}	0.53	0.43	0.46	0.53	0.60	0.54	0.58	0.60
	KL_{CT}	0.50	0.49	0.52	0.53	0.60	0.48	0.58	0.60
	AVG_{Base}	0.53	0.52	0.51	0.53	0.56	0.61	0.64	0.60
	$AVG_{PT\text{-}Rank}$	0.53	0.62	0.54	0.53	0.60	0.53	0.56	0.60
	$AVG_{CT\text{-}Rank}$	0.53	0.50	0.44	0.53	0.60	0.58	0.57	0.60
	$JS_{PT\text{-}Rank}$	0.50	0.52	0.47	0.49	0.64	0.67	0.69	0.53
	$JS_{CT\text{-}Rank}$	0.51	0.57	0.48	0.46	0.59	0.55	0.60	0.60
	$Clarity+KL_{PT}+KL_{CT}$	0.54	0.42	0.42	0.54	0.61	0.50	0.46	0.61
	$Clarity+JS_{PT\text{-}Rank}+JS_{CT\text{-}Rank}$	0.58	0.42	0.46	0.54	0.61	0.54	0.57	0.61
	ALL	0.51	0.37	0.26	0.53	0.57	0.48	0.30	0.60

Table 7.3: Retrieval effectiveness of different ranking methods measured using P@1, P@5, P@10 and MAP; * indicates statistically different from *CT-Rank* using t-test with significant at $p < 0.05$.

Method	*exclusive*				*inclusive*			
	P@1	*P@5*	*P@10*	*MAP*	*P@1*	*P@5*	*P@10*	*MAP*
CT-Rank	0.55	0.50	0.48	0.53	0.58	0.55	0.53	0.56
PT-Rank	0.63*	0.53	0.50	0.55	0.63	0.58	0.55	0.61
Rank-Prediction	0.68*	0.53	0.50	0.59*	0.70*	0.58	0.59	0.64*
MAX	0.83*	0.61*	0.52	0.64*	0.78*	0.62*	0.59	0.67*

7.6 Conclusions

In this chapter, we studied and compared time-aware ranking models based on two time dimensions: publication time and content time. We demonstrated that temporal queries can benefit from different ranking models, that is, the retrieval effectiveness differs among ranking models. According to this, we categorized queries based on two time dimensions, and we proposed to predict the suitable ranking model using supervised machine learning. In order to evaluate our approach, we conducted extensive experiments using temporal queries and relevance judgment using crowdsourcing. The results show that our prediction-based ranking model outperforms the baseline significantly.

It is obvious that when comparing with the optimal case there is still room for further improvements. In future work we plan to increase the accuracy of ranking prediction by studying additional features.

Part IV

Retrieval and Ranking Models

Chapter 8

Comparison of Time-aware Ranking Methods

In general, a time-aware ranking method ranks documents that are textually and temporally similar to a query and ranks retrieved documents with respect to both similarities. Previous work has followed one of two main approaches: 1) a mixture model linearly combining textual similarity and temporal similarity, or 2) a probabilistic model generating a query from the textual and temporal part of a document independently. In this chapter, we address the research question: *how to explicitly model the time dimension into retrieval and ranking?*, by performing an empirical study and evaluation of different time-aware ranking methods using the same dataset.

8.1 Motivation

The previous time-aware ranking methods [10, 31, 54, 74] are based on two main approaches: 1) a mixture model linearly combining textual and temporal similarity, or 2) a probabilistic model generating a query from the textual and temporal part of a document independently. It is shown that time-aware ranking performs better than keyword-based ranking, e.g., tf-idf and language modeling. To the best of our knowledge, an empirical comparison of different time-aware ranking methods using the same dataset has never been done before.

Contributions

Our main contributions in this chapter are as follows.

- We perform the first study and analysis of different time-aware ranking methods.
- By conducting extensive experiments, we compare the performance of different time-aware ranking methods using the same dataset.

Organization

The organization of the rest of the chapter is as follows. In Section 8.2, we give an overview of related work. In Section 8.3, we first outline the models for documents and queries, and we present a mixture model of time-aware ranking. In Section 8.4, we describe different time-aware ranking methods, and we conduct extensive experiments in order to evaluate different time-aware ranking methods in Section 8.5. Finally, in Section 8.6, we conclude the chapter.

8.2 Related Work

In this section, we give an overview of ranking methods that incorporate temporal information, and point out their underlying aspects including: 1) explicit or implicit temporal information needs, 2) uncertainty-concern or uncertainty-ignore, and 3) using timestamps or temporal expressions.

A number of ranking models exploiting temporal information have been proposed, including [3, 7, 31, 54, 74, 87]. In [74], Li and Croft incorporated time into language models, called time-based language models, by assigning a document prior using an exponential decay function of the publication time of document, i.e., the creation date. They did not have temporal information needs explicitly provided, but they focused on recency queries. The time uncertainty is captured by the exponential decay function, such that the more recent documents obtain the higher probabilities of relevance.

In [31], Diaz and Jones measure the distribution of creation dates of retrieved documents to create the temporal profile of a query. The temporal profile was presented due to no explicit temporal information needs. Hence, they needed to estimate the time relevant to a query by analyzing the distribution of creation dates. Their results showed that the temporal profile together with the contents of retrieved documents can improve averaged precision for the query by using a set of different features for discriminating between temporal profiles.

In [54], Kalczynski and Chou proposed a temporal retrieval model for news archives. In their work, temporal expressions in a query and documents were explicitly modeled in ranking. A query is defined as a set of precise temporal information needs, i.e., the finest time chronon, or a day. Thus, they assumed that the uncertainty applied only to temporal references in documents, and it was represented as a fuzzy set function.

The work by Baeza-Yates [7] proposed to extract temporal expressions from news, index news articles together with temporal expressions, and retrieve temporal information (in this case, future-related events) by using a probabilistic model. A document score is given by multiplying a *keyword* similarity and a time confidence, i.e., a probability that the document's events will actually happen. We can view the confidence as the uncertainty of time. Besides, this work allowed a user to explicitly specify temporal information needs, but only on a year-level granularity.

Metzler et al. [87] considered implicit temporal information needs. They proposed mining query logs and analyze query frequencies over time in order to identify strongly time-related queries. They did not directly extract temporal expressions from queries and

documents. In addition, they presented a ranking model concerning implicit temporal needs, and the experimental results showed the improvement of the retrieval effectiveness of temporal queries for web search.

In more recent work, Berberich et al. [10] integrated temporal expressions into query-likelihood language modeling, which considers uncertainty inherent to temporal expressions in a query and documents. That is, temporal expressions can refer to the same time interval even they are not exactly equal. The work by Berberich et al. required explicit temporal information needs as a part of query.

We will later detail different time-aware ranking methods: LMT [10], LMTU [10], TS(cf. Chapter 4), TSU(cf. Chapter 4), and FuzzySet [54] that underline different aspects of time and uncertainty in Section 8.4.

8.3 Models for Documents and Queries

A temporal query q is composed of keywords q_{text} and temporal expressions q_{time}. A document d consists of the textual part d_{text}, i.e., a bag of words, and the temporal part d_{time} composed of the publication date $PubTime(d)$, and temporal expressions $\{t_1, \ldots, t_k\}$ mentioned in the document's contents $ContentTime(d)$. Both the publication date and temporal expressions will be represented using the time model of Berberich et al. [10] presented in Section 2.2.2.

8.4 Time-aware Ranking Methods

We study different time-aware ranking methods proposed to measure temporal similarity between a query and a document including: LMT [10], LMTU [10], TS (cf. Chapter 4), TSU (cf. Chapter 4), and FuzzySet [54]. Although they are shown the good performance in the retrieval of temporal needs, those methods have never been compared using the same dataset and relevance judgments. In the following, we describe in detail each time-aware ranking method. The summarization of characteristics of the time-aware ranking methods with respect to two aspects is shown in Table 8.1.

Table 8.1: Characteristics of different time-aware ranking models.

Method	Time		Uncertainty	
	Publication	*Content*	*Ignore*	*Concern*
LMT	x	√	√	x
LMTU	x	√	x	√
TS	√	x	√	x
TSU	x	√	x	√
FuzzySet	√	x	x	√

To be comparable, we apply a mixture model to linearly combine textual similarity and temporal similarity for all ranking methods. Given a temporal query q, a document d

will be ranked according to a score computed as follows:

$$S(q,d) = (1-\alpha) \cdot S'(q_{text}, d_{text}) + \alpha \cdot S''(q_{time}, d_{time}) \tag{8.1}$$

where the mixture parameter α indicates the importance of textual similarity $S'(q_{text}, d_{text})$ and temporal similarity $S''(q_{time}, d_{time})$. Both similarity scores must be normalized, e.g., divided by the maximum scores, in order to the final score $S(q,d)$. $S'(q_{text}, d_{text})$ can be measured using any of existing text-based weighting functions. $S''(q_{time}, d_{time})$ measure temporal similarity by assuming that a temporal expression $t_q \in q_{time}$ is generated independently from each other, and a two-step generative model was used [10]:

$$\begin{aligned} S''(q_{time}, d_{time}) &= \prod_{t_q \in q_{time}} P(t_q|d_{time}) \\ &= \prod_{t_q \in q_{time}} \left(\frac{1}{|d_{time}|} \sum_{t_d \in d_{time}} P(t_q|t_d) \right) \end{aligned} \tag{8.2}$$

Linear interpolation smoothing will be applied to give the probability $P(t_q|t_d)$ for an unseen query temporal expression t_q in d. In the next section, we will explain how to estimate $P(t_q|t_d)$ for different time-aware ranking methods.

The temporal ranking methods LMT and LMTU are based on a generative model approach. Similar to a query-likelihood approach, the textual and temporal part of the query q are generated independently from the corresponding parts of the document d as:

$$P(q|d) = P(q_{text}|d_{text}) \times P(q_{time}|d_{time}) \tag{8.3}$$

The textual similarity part $P(q_{text}|d_{text})$ can be determined by an existing text-based query-likelihood approach, e.g., the original Ponte and Croft model [101].

A temporal expression q_{time} are assumed to be generated independently from each other. To generate each temporal expression t_q in q_{time} from d, a two-step generative model was used. First, a document temporal expression t_d is drawn at uniform random from document temporal expressions d_{time}. Second, a query temporal expression t_q in q_{time} is generated from a temporal expression t_d in d.

$$\begin{aligned} P(q_{time}|d_{time}) &= \prod_{t_q \in q_{time}} P(t_q|d_{time}) \\ &= \prod_{t_q \in q_{time}} \left(\frac{1}{|d_{time}|} \sum_{t_d \in d_{time}} P(t_q|t_d) \right) \end{aligned} \tag{8.4}$$

The probability of generating t_q from t_d or $P(t_q|t_d)$ can be calculated using two different methods: LMT and LMTU. The first method ignores the uncertainty, i.e., only temporal

expressions are exactly equal will be considered. Thus, $P(t_q|t_d)$ under LMT can be computed as:

$$P(t_q|t_d)_{LMT} = \begin{cases} 0 & \text{if } t_q \neq t_d, \\ 1 & \text{if } t_q = t_d. \end{cases} \tag{8.5}$$

Contrary to LMT, the ranking method LMTU takes the uncertainty into account, i.e, it assumes equal likelihood for each time interval t'_q that t_q can refer to. More precisely, a set of time intervals $t_q = \{t'_q | t'_q \in t_q\}$ that the user may have had in mind when issuing the query are assumed equally likely. Recall that the number of time intervals in t_q, denoted $|t_q|$, can be very huge. $P(t_q|t_d)$ under LMTU can be calculated as:

$$P(t_q|t_d)_{LMTU} = \frac{1}{|t_q|} \sum_{t'_q \in t_q} P(t'_q|t_d) \tag{8.6}$$

$$P(t'_q|t_d) = \frac{1}{|t_d|} \begin{cases} 0 & \text{if } t'_q \notin t_d, \\ 1 & \text{if } t'_q \in t_d. \end{cases} \tag{8.7}$$

Finally, the simplified calculation of $P(t_q|t_d)$ is given as follows.

$$P(t_q|t_d)_{LMTU} = \frac{|t_q \cap t_d|}{|t_q| \cdot |t_d|} \tag{8.8}$$

As explained in [10], |t| can be computed efficiently for any content time or temporal expression t in two cases as:
(1) if $tb_u \leq te_l$ then |t| can simply be computed as:

$$|t| = (tb_u - tb_l + 1) \cdot (te_u - te_l + 1)$$

(2) if $tb_u > te_l$ then |t| can be computed as:

$$\begin{aligned} |t| &= \sum_{tb=tb_l}^{tb_u} (te_u - max(tb, te_l) + 1) \\ &= (te_l - tb_l + 1) \cdot (te_u - te_l + 1) \\ &+ (tb_u - te_l) \cdot (te_u - te_l + 1) - 0.5 \cdot (tb_u - te_l) \cdot (tb_u - te_l + 1) \end{aligned}$$

Note that, $P(t_q|t_d)$ for both LMT and LMTU methods is prone to the zero-probability problem. Thus, Jelinek-Mercer smoothing is applied, and the estimated value $\hat{P}(t_q|t_d)$ becomes:

$$\hat{P}(t_q|t_d) = (1 - \lambda_1) \cdot \frac{1}{|C_{time}|} \sum_{t_d \in C_{time}} P(t_q|t_d) + \lambda_1 \cdot \frac{1}{|d_{time}|} \sum_{t_d \in d_{time}} P(t_q|t_d) \tag{8.9}$$

where the smoothing parameter $\lambda_1 \in [0, 1]$, and C is the whole document collection.

In Chapter 4, we proposed to measure the temporal similarity using TS and TSU. Instead of using a language modeling approach as in [10], we employed a mixture model approach to combining the time similarity with the textual similarity. The mixture model-based approach is given as:

$$S(q,d) = (1-\alpha)\cdot S'(q_{text}, d_{text}) + \alpha \cdot S''(q_{time}, d_{time}) \tag{8.10}$$

where α is a parameter underlining the importance of both similarity scores: textual similarity $S'(q_{text}, d_{text})$ and temporal similarity $S'(q_{time}, d_{time})$. The textual similarity can be implemented using an existing text-based weighting models, e.g. tf-idf. The value of textual similarity must be normalized using the maximum keyword score among all documents as:

$$S'_{norm}(q_{text}, d_{text}) = \frac{S'(q_{text}, d_{text})}{\max S'(q_{text}, d_{text})} \tag{8.11}$$

$S''(q_{time}, d_{time})$ or the temporal similarity part is defined using two methods: TS and TSU. Both methods ignore temporal expressions in documents, that is, they represented d using the creation date only, and d_{time} is referred to $\mathit{PubTime}(d)$.

The probability of generating $q_{time}$from d_{time}, or $S''(q_{time}, d_{time})$ can be computed as:

$$\begin{aligned} S''(q_{time}, d_{time}) &= P(q_{time}|d_{time}) \\ &= \frac{1}{|q_{time}|}\sum_{t_q \in q_{time}} P(t_q|d_{time}) \end{aligned} \tag{8.12}$$

where q_{time} is a set of query temporal expressions. Hence, $P(q_{time}|d_{time})$ is averaged over the probability of generating each temporal expression in q_{time}, or $P(t_q|d_{time})$.

Similar to LmT and LmTU, the probability of generating a time interval t_q given d_{time} (i.e., $\mathit{PubTime}(d)$) can be calculated in two ways: 1) ignoring uncertainty, and 2) taking uncertainty into account. By ignoring uncertainty, $P(t_q|d_{time})$ is defined as:

$$P(t_q|d_{time})_{TS} = \begin{cases} 0 & \text{if } \mathit{PubTime}(d) \notin t_q, \\ 1 & \text{if } \mathit{PubTime}(d) \in t_q. \end{cases} \tag{8.13}$$

In this case, the probability of generating a query temporal expression is equal to 1 only if the publication date of d is in a range of t_q, or it is equal 0 otherwise. In the case where uncertainty is concerned, $P(t_q|d_{time})$ is defined using an exponential decay function:

$$P(t_q|d_{time})_{TSU} = DecayRate^{\lambda_2 \cdot |t_q - t_d|} \tag{8.14}$$

$$|t_q - t_d| = \frac{|tb_l^q - tb_l^d| + |tb_u^q - tb_u^d| + |te_l^q - te_l^d| + |te_u^q - te_u^d|}{4} \tag{8.15}$$

where $t_d = \mathit{PubTime}(d)$, $DecayRate$ and λ are constant, $0 < DecayRate < 1$ and $\lambda > 0$, and μ is a unit of time distance. Intuitively, this function gives a probability that

decreases proportional to the difference between a time interval t_q and the publication date of d. A document with its creation date closer to t_q will receive a higher probability than a document with its creation date farther from t_q. Note that, LMTU concerns the uncertainty by exploiting *all possible time interval* inherent in a temporal expression into the calculation, whereas TSU ignore this assumption but TSU concerns the uncertainty by taking account of *a time distance* (i.e., measuring by a decay function) between two time intervals.

The normalization of $S''_{norm}(q_{time}, d_{time})$ can be computed in two ways:

1. uncertainty-ignorant using $P(t_q|d_{time})_{TS}$ defined in Equation 8.13
2. uncertainty-aware using $P(t_q|d_{time})_{TSU}$ defined in Equation 8.14

Finally, the normalized value of $S''_{norm}(q_{time}, d_{time})$ will be substituted $S''(q_{time}, d_{time})$ in Equation 8.10 yielding the normalized score of a document d given a temporal query q with determined time q_{time} as follows:

$$S_{norm}(q, d) = (1 - \alpha) \cdot S'_{norm}(q_{text}, d_{text}) + \alpha \cdot S''_{norm}(q_{time}, d_{time}) \tag{8.16}$$

Kalczynski and Chou [54] measured the temporal similarity between a query and a document using a fuzzy membership function with *t* different shapes, so-called *t*-zoidal. Rather than assuming all time intervals inherent in a temporal expression, they propose to capture the uncertainty of time using the fuzzy membership function. In this work, we only consider the *6*-zoidal fuzzy membership function illustrated in Figure 8.1.

The figure depicts a query temporal expression $t_q = [t_a, t_b]$ with the beginning point t_a and the ending point t_b equivalent to the points a_2 and a_3 respectively. The time of document d_{time} can be any point on a timeline. The temporal similarity between q and d will be computed based on the graphical function in this figure. In addition, this method also ignores temporal expressions in documents, i.e., they represented d using the creation date only, and d_{time} is referred to *PubTime*(d). Thus, *FuzzySet* is defined as:

$$\textit{FuzzySet} = \begin{cases} 0 & \text{if } t_d < a_1, \\ f_1(t_d) & \text{if } t_d \geq a_1 \wedge t_d \leq a_2, \\ 1 & \text{if } t_d > a_2 \wedge t_d \leq a_3, \\ f_2(t_d) & \text{if } t_d > a_3 \wedge t_d \leq a_4, \\ 0 & \text{if } t_d > a_4. \end{cases} \tag{8.17}$$

$$f_1(t_d) = \begin{cases} \left(\frac{a_1 - t_d}{a_1 - a_2}\right)^n & \text{if } a_1 \neq a_2, \\ 1 & \text{if } a_1 = a_2. \end{cases} \tag{8.18}$$

$$f_2(t_d) = \begin{cases} \left(\frac{a_4 - t_d}{a_4 - a_3}\right)^m & \text{if } a_3 \neq a_4, \\ 1 & \text{if } a_3 = a_4. \end{cases} \tag{8.19}$$

where t_d is equal to d_{time}. The parameters a_1, a_4, n, m will be determined empirically.

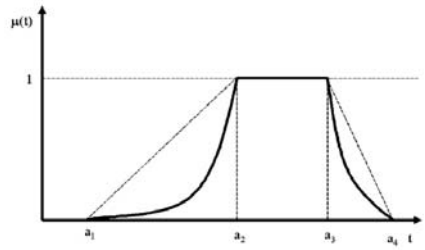


Figure 8.1: The 6-zoidal membership function from [54].

Finally, we will conclude the similarity/dissimilarity of different temporal ranking methods we presents. In other words, we want to remark the difference among them using the following metrics: uncertainty-ignorance or uncertainty-concern, and whether exploiting temporal expressions in either a query or a document, or both.

8.5 Evaluation

We first describe the settings of experiments. Then, we evaluate different time-aware ranking methods, and discuss the results.

8.5.1 Setting

Temporal document collection. We used the New York Times Annotated Corpus as a temporal document collection. Note that, the proposed ranking is not limited this particular collection, but it can be applied to other temporal collections as well. The Apache Lucene search engine version 2.9.3 was used for indexing/retrieving documents.

Queries and relevance assessments. In this work, a standard query and relevance judgment benchmark, such as, TREC, is not useful because queries are not time-related, and the judgment is not targeted towards temporal information needs. For this reason, we used the same set of queries and relevance assessments as the work by Berberich et al [10]. They obtained 40 temporal queries and 6,255 query/document judgments using 5 assessors from the Amazon Mechanical Turk (AMT).

Document annotation. To extract features from the New York Times Annotated Corpus, a series of language processing tools were used as described in [85], including OpenNLP [97] (for tokenization, sentence splitting and part-of-speech tagging, and shallow parsing), the SuperSense tagger [114] (for named entity recognition) and TARSQI Toolkit [119] (for annotating documents with TimeML and extracting temporal expressions). The result of this analysis were: 1) entity information, e.g., all of persons, locations and organizations, 2) temporal expressions, e.g., all of event dates, and 3) sentence information, e.g., all sentences, entities and event dates occurs in each sentence, as well as position information.

Parameter setting. The smoothing parameter was set to 0.1. Parameters for TSU were: $DecayRate = 0.5$, $\lambda = 0.5$, and $\mu = 6$ months. Parameters for FuzzySet were $n = 2$, $m = 2$, $a_1 = a_2 - (0.25 \times (a_3 - a_2))$, and $a_4 = a_3 + (0.50 \times (a_3 - a_2))$.

Evaluating an individual ranking method. To compare different methods, we used a mixture model, where the Lucene's default weighting function was used to capture the textual similarity for all ranking methods. In this way, the results of each temporal ranking can be comparable. The mixture parameter α was varied in the experiments. Each retrieved document is ranked with respect to $S(q, d)$ in Equation 8.16, where $S'(q_{text}, d_{text})$ was a score obtained from the Lucene's default weighting function, and $S''(q_{time}, d_{time})$ was obtained from different time-aware ranking methods described in Section 8.4. The baseline was the textual similarity $S'(q_{text}, d_{text})$, i.e., the Lucene's default weighting function, using *inclusive* mode denoted TFIDF-IN. These two retrieval modes were applied to each temporal ranking method, and the results will be reported accordingly.

Metrics. The retrieval effectiveness of temporal ranking was measured by the precision at 1, 3, 5 and 10 documents (P@1, P@3, P@5 and P@10 respectively), Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP). For the learned ranking method, the average performance over the five folds was used to measure the overall performance of each ranking model.

8.5.2 Results

First, we study the sensitivity of each temporal ranking method to the mixture parameter α. The effectiveness (P@5, P@10, and MAP) of each temporal ranking method when varying α. For *inclusive mode*, the sensitivity of each temporal ranking method is shown in Figure 8.2. For *exclusive mode*, the sensitivity of each temporal ranking method is shown in Figure 8.2. Note that, suffixes IN and EX refer to *inclusive* and *exclusive* mode respectively. Next, we will compare different ranking methods using the best performed results with respect to this sensitivity.

The effectiveness of the baseline (i.e., the Lucene's default weighting function) and different temporal ranking methods are displayed in Table 8.2. In general, the exclusive mode performed better than the inclusive mode for both LMT and LMTU, and LMTU-EX gained the best performance over the other baselines.

Table 8.2 shows the best performing results of each method. In general, all time-aware ranking methods outperform the baseline significantly, except LMT. For each time-aware ranking, the effectiveness when retrieved using *exclusive* is better than *inclusive*. TSU performs best among all methods in both *inclusive* and *exclusive* modes, and it outperforms all other methods significantly for P@1, MAP and MRR.

8.6 Conclusions

Time-aware ranking methods show better performance compared to methods based on keywords only. When the time-uncertainty is taken into account, the effectiveness is improved significantly. Even though TSU gains the best performance among other methods,

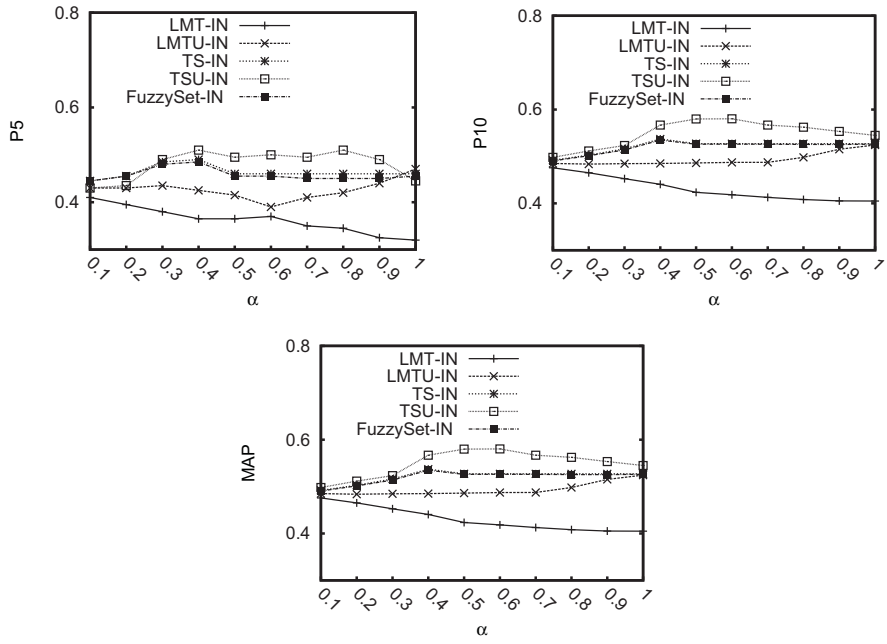


Figure 8.2: Sensitivity of P@5, P@10 and MAP to the mixture parameter α for *inclusive* mode.

the usefulness of TSU is still limited for a document collection with no time metadata, i.e., the publication time of documents is not available. On the contrary, LMT and LMTU can be applied to any document collection without time metadata, but extraction of temporal expressions is needed.

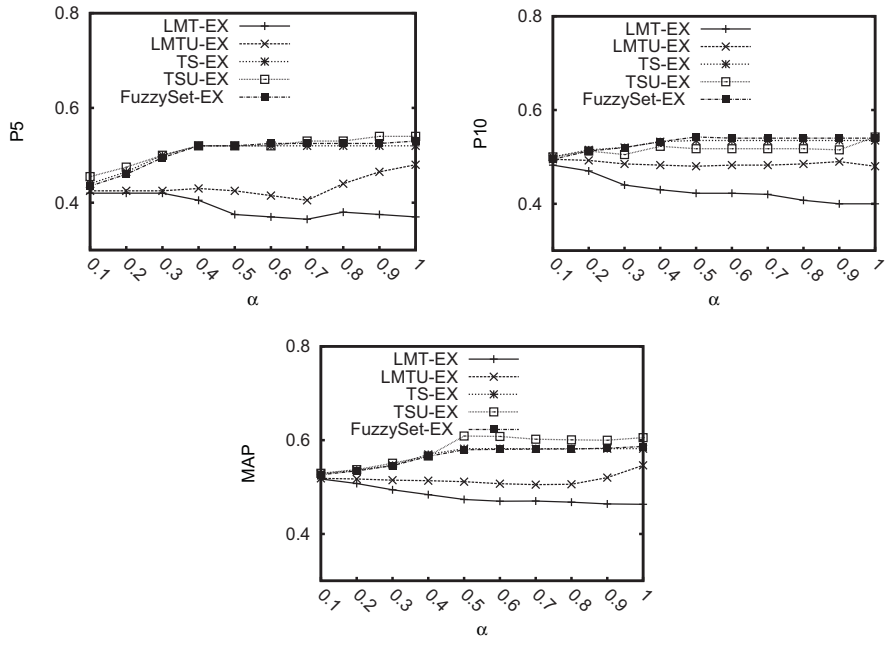


Figure 8.3: Sensitivity of P@5, P@10 and MAP to the mixture parameter α for *exclusive* mode.

Table 8.2: Effectiveness of different time-aware ranking methods (suffixes IN and EX refer to *inclusive* and *exclusive* mode respectively), * indicates statistically improvement over the baselines using t-test with significant at $p < 0.05$.

Methods	P@1	P@3	P@5	P@10	MAP	MRR
TFIDF-IN	.38	.45	.43	.41	.49	.56
LMT-IN	.43	.43	.41	.41	.48	.57
LMTU-IN	.48	.49	.47	.45	.52	.68
TS-IN	.45	.48	.49	.48	.54	.61
TSU-IN	**.65**	.56	.51	.49	**.58**	**.76**
FuzzySet-IN	.45	.48	.49	.48	.53	.61
LMT-EX	.38	.46	.42	.48	.52	.55
LMTU-EX	.48	.52	.48	.50	.55	.68
TS-EX	.48	.56	.52	.53	.58	.63
TSU-EX	**.68**	.58	.54	.54	**.61**	**.77**
FuzzySet-EX	.48	.55	.53	.54	.59	.64

Chapter 9

Ranking Related News Predictions

In the previous chapter, we presented different ranking models that retrieve documents with respect to textual and temporal similarity. In this chapter, we also want to investigate whether exploiting other features together with time can help improving the retrieval effectiveness in searching temporal document collections. Specifically, we set up a new task called *ranking related news predictions*, which is aimed at retrieving and ranking sentences that contain mentions to future events. The research question addressed in this chapter is: *how to combine different features with time in order to improve relevance ranking?*

9.1 Motivation

Predicting the future has long been the Holy Grail in the financial world. The leaders of large organizations need to analyze information related to the future in order to identify the key challenges that can directly affect their organizations. This information can be useful for strategies planning to avoid/minimize disruptions, risks, and threats, or to maximize new opportunities [19]. For example, a business company usually concerns about clients' interests in global competition, innovation and profits, and a government's challenges are in areas of education, energy, security and health care. However, it is not just businesses that care about the future - all people have anticipation and curiosity about the future. Canton [19] describes the future trends that can influence our lives, our jobs, our businesses, and even our world. These include the energy crisis, the global financial crisis, politics, health care, science, securities, globalization, climate changes, and technologies. When people read news stories on any of these topics whether it is an article about war in the Middle East or the latest health care plan, they are naturally curious about potential future events. How long will the war last? How much will it cost? What happens if we do nothing at all? This obsession with the future is also reflected in the news articles themselves - our analysis of one year worth of news from over 100 sources indicates that nearly one third of news articles contain at least one statement made about a future date.

Accessing this information in an intuitive way would greatly improve how people

read and understand news. In this chapter, we define a new task we call *ranking related news predictions* that directly addresses this problem by finding all predictions related to a news story in a news archive and ranking them according to their relevance to the news story. This task is motivated by the desire of news sites to increase user engagement by providing content that directly addresses the information needs of users. By providing links to relevant content, new sites can keep users on their site longer thus increasing the likelihood that users will click on revenue generating links and also improving user satisfaction. For a wide range of news events from natural disasters to political unrest in the Middle East, the information need - the question most on people's minds - is what is going to happen next. This new task is a first step toward helping people answer this very question by finding and linking to predictions that are relevant to the user.

Our query is extracted from a news article currently read by a user, and is composed of a bag of *entities* or *terms*. Using an automatically-generated query, predictions are retrieved, ranked over the time dimension, and presented to the user. Note that there are a number of future-related information analyzing tools including *Recorded Future* [102], and *Time Explorer* [85]. *Recorded Future* extracts predictions from different sources (news publications, blogs, trade publications, government web sites, and financial databases). A user creates a query by selecting a topic of interest (e.g. a topic about "Financial Markets"), and then specifying an entity (people, companies, or organizations) from a set of "predefined" entities. The system will then retrieve predictions related to the selected topic and entity. A major difference with our system is that *Recorded Future* requires a query specified in advance, while our system automatically creates a query for the user based on the news article being read and it is not limited to "predefined" entities. Besides, *Recorded Future* lacks of the ranking of predictions whereas we rank predictions before presenting them to the user. *Time Explorer* is a search engine that allows users to see how topics have evolved over time and how they might continue to evolve in the future. The system extracts predictions from document collections and allows users to search for them using ad-hoc queries. However, neither *Time Explorer* nor *Recorded Future* provide details of how predictions are ranked nor do they evaluate performance in a formal setting as we do here. However, there is no ranking of predictions in *Time Explorer* as we will do in this work.

In this chapter, we will propose a ranking model of future information using machine learning techniques. To learn the ranking model, we define 4 classes of features to measure different similarities, namely, term similarity, semantic similarity, topic similarity, and temporal similarity. These features are aimed at capturing the similarity between an information need and predictions of future-related events. In addition, we explicitly exploit temporal information of a query and documents (i.e. temporal expressions) in ranking. The challenges of our *ranking related news predictions* task are related to various aspects of IR problems: sentence retrieval, entity ranking, temporal ranking, and domain-specific predictions.

Contributions

The main contributions of this chapter are as follows.

- We propose the first formalization of the *ranking related news predictions* task.
- We propose a learned ranking model incorporating four classes of features including term similarity, entity-based similarity, topic similarity, and temporal similarity.

Organization

The organization of the rest of the chapter is as follows. In Section 9.2, we give an overview of related work. In Section 9.3, we explain our system architecture, and outline the models for annotated documents, predictions as well as queries. In Section 9.4, we propose four classes of features used for learning a ranking model. In Section 9.5, we describe our ranking model. In Section 9.6, we evaluate the proposed ranking model. Finally, in Section 9.7, we conclude our work in this chapter.

9.2 Related Work

Our related work includes sentence retrieval, entity ranking, temporal ranking, and domain-specific predictions.

Sentence retrieval is the task of retrieving a relevant sentence related to a query. Different application areas of sentence retrieval are mentioned in the book of Murdock [90] and references therein, including, for example, question answering [115], text summarization, and novelty detection. Surdeanu et al. [115] applied supervised learning to rank a set of short answers (sentences) matched a given question (query) by using different classes features. Li and Croft [75] proposed to detect novelty topics by analyzing sentence-level information (sentence lengths, named entities, and opinion patterns). Generally, because sentences are much smaller than documents and thus have limited content compared to documents, the effectiveness of the retrieval of sentences is significantly worse. To address this problem, Blanco and Zaragoza [15] proposed to use the context of sentences in order to improve the effectiveness of sentence retrieval.

There have been a number shared tasks with the goal of furthering research in the area of entity ranking. For instance, the TREC 2008 Enterprise track was created with the objective to find experts (or people) related to a given topic of interest. The INEX Entity Ranking track [30] was launched with the task of finding a list of relevant entities (represented by Wikipedia articles) for a given topic. Recently, the TREC 2009 Entity track was introduced, and the task is to find related entities (represented by homepages) given a topic (called a source entity). The difference between the TREC 2009 Entity and the previous tracks is that it allows a relation and a target entity type to be explicitly specified. There are various approaches to ranking entities by using language models [9], voting models [81], and entity-based graph models [137].

Many ranking models exploiting temporal information have been proposed, including [10, 31, 74, 87]. Li and Croft [74] experimented with time-based language models by assigning a document prior using an exponential decay function of its creation date, such that the more recent documents obtain the higher probabilities of relevance. Diaz and

Jones [31] build a temporal profile of a query from the distribution of document publication dates. They use time dependent features derived from these profiles that improve the ranking of temporal queries. In [40], Gwadera and Crestani proposed a method for mining and ranking news stories using cross-stream sequential patterns and content similarity.

Berberich et al. [10] integrated temporal expressions into query-likelihood language modeling, which considers uncertainty inherent to temporal expressions in a query and in documents, i.e., two temporal expressions can refer to the same time interval even when they are not exactly equal. Metzler et al. [87] mined query logs to identify implicit temporal information needs and presented a time-dependent ranking model for certain types of queries. Elsas and Dumais [33] also take time into retrieval. They demonstrate that the relevance of a document is strongly correlated with its content change frequency.

There is much research in domain-specific predictions such as stock market predictions [109, 132] and recommender systems [72, 98]. The first aims at predicting stock price movements by analyzing financial news, while the latter applies collaborative filtering algorithms for recommending books, videos, movie, etc. based on users' interests.

The future retrieval problem was first presented by Baeza-Yates [7]. He proposed to extract temporal expressions from news, index news articles together with temporal expressions, and retrieve future information (composed of text and future dates) by using a probabilistic model. A document score is given as a multiplication of a *keyword* similarity and a time confidence, i.e., a probability that the document's events will actually happen. The limitation of this original work is that it is evaluated using a small data set and only a year granularity is used.

The more recent work on the future-related information retrieval is presented by Jatowt et al. [50]. In contrast to our work, they do not focus on relevance and ranking future-related information retrieval. They presented an analytical tool for extracting, summarizing and aggregating future-related events from news archives, but did not perform an extensive evaluation, only calculating averaged precision on a small set of generated results.

9.3 Problem Definition

In this section, we outline the system architecture, and give the formalization of the models for annotated documents, predictions, and queries.

9.3.1 System Architecture

Figure 9.1 depicts our system which retrieves a set of predictions (sentences containing future dates) related to a given news article. Predictions can be extracted from a temporal document collection – any collection that contains timestamped documents, e.g., personal emails, news archives, company websites and blogs. In this work, we automatically extract predictions from news archives using different annotation tools. Our *document annotation process* includes tokenization, sentence extraction, part-of-speech tagging, named entity recognition, and temporal expression extraction. The result of this

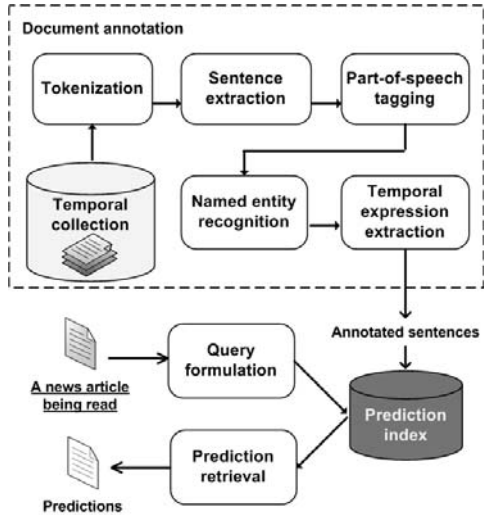


Figure 9.1: Prediction retrieval system architecture.

process is a set of sentences annotated with named entities and temporal expressions, which will be indexed as *predictions* for further processing or retrieval.

A key component of the annotation process is the extraction of temporal expressions using a time and event recognition algorithm. The algorithm extracts temporal expressions mentioned in a document and normalizes them to dates so they can be anchored on a timeline. Instead of having an explicit information need provided, we automatically generate a query. In this case, we assume that the user's information needs lie in *the news article being read* by the user, and a query will be extracted from this news article (further details are given in Section 9.3.4). For a given news article, we retrieve predictions that are relevant to the news article, that is, relevant sentences containing future dates with respect to the publication date of the news article being read.

Retrieved predictions are ranked by the degree of relevance, where a prediction is "relevant" if it is future information about *the topics of the news article*. Note that we do not give any specific instructions about how the dates involved are related to relevance. Nevertheless, we hypothesize that predictions extracted from more recent documents are more relevant. In this chapter, we use a machine learning approach [77] for learning the ranking model of predictions. This involves identifying different classes of features (see Section 9.4) to measure the relevance of a prediction with respect to the news article.

9.3.2 Annotated Document Model

Our document collection contains a number of news articles defined as $C = \{d_1, \ldots, d_n\}$. We treat each news article as a bag-of-words (an unordered list of terms, or features), $d = \{w_1, \ldots, w_n\}$. $\mathit{time}(d)$ is a function given the creation or publication date of d. Some of our proposed features are extracted from annotated documents, which are defined as

Table 9.1: Example of a prediction with field/value pairs.

Field	Value
ID	1136243_1
PARENT_ID	1136243
TITLE	*Gore Pledges A Health Plan For Every Child*
TEXT	*Vice President* **Al Gore** *proposed today to guarantee access to affordable health insurance for all children by 2005, expanding on a program enacted two years ago that he conceded had had limited success so far.*
CONTEXT	*Mr. Gore acknowledged that the number of Americans without health coverage had increased steadily since he and President Clinton took office.*
ENTITY	Al Gore
FUTURE_DATE	2005
PUB_DATE	1999/09/08

follows. Each document d, has an associated annotated document $\hat{d}$, which will consist of three sets, $\hat{d}_e$, $\hat{d}_t$, $\hat{d}_s$: a set of named entities $\hat{d}_e = \{e_1, \ldots, e_n\}$, where each entity $e_i \in \mathcal{E}$ and $\mathcal{E}$ is the complete set of entities (typed as person, location, and organization) in the collection; a set of annotated temporal expressions $\hat{d}_t = \{t_1, \ldots, t_m\}$ and a set of sentences $\hat{d}_s = \{s_1, \ldots, s_z\}$

9.3.3 Prediction Model

A prediction p can be viewed as a sentence containing field/value pairs of annotation information and we define d^p as the *parent* document where p is extracted from. We define several fields for a prediction including ID, PARENT_ID, TITLE, ENTITY, FUTURE_DATE, PUB_DATE, TEXT, and CONTEXT. The field ID specifies a prediction's unique number, PARENTID and TITLE represent a unique number and the title of d^p respectively ENTITY contains a set of annotated entities $p_{entity} \subset \hat{d}_e$, FUTURE_DATE consists of "future" temporal expressions p_{future} annotated in p, PUB_DATE is the publication date of the *parent* document d^p and TEXT is a prediction's text p_{txt} or the sentence of p. Note that each prediction must contain at least one "future" temporal expression, that is, $p_{future} \neq \emptyset$. In addition, we explicitly model the context of the prediction p_{ctx}, represented by the field CONTEXT and defined as surrounding sentences of the main sentence [15]. In our work, we define the context p_{ctx} as the sentence immediately before and the one immediately after p_{txt}. Table 9.1 contains an example of a prediction with its field/value pairs.

9.3.4 Query Model

As mentioned earlier, a query q is automatically generated from a news article being read d^q; q is composed of two parts: keywords q_{text}, and the time of query q_{time}. The keywords q_{text} are extracted from d^q in three ways resulting in three different types of queries.

The first type of query is called "entity query", and it is defined as follows.

Definition 12 (Entity Query)**.** *An entity query is represented by a bag of entities denoted Q_E. The keyword part q_{text} of Q_E is composed of top-m entities ranked by frequency, where the entities are extracted from d^q.*

Intuitively, we want to know whether using only *key* entities frequently mentioned in the news article can retrieve relevant predictions with high precision or not. For example, given an actual document about "President Bush and the Iraq war", we extract Q_E with $q_{text} = \langle \textit{George Bush}, \textit{Iraq}, \textit{America} \rangle$. At retrieval time, q_{text} will be matched with the ENTITY field of the predictions. Note that, the frequency of entities will be considered only for selecting the *top-m* entities but it will not be used for retrieval.

The second query is called "term query" and it is defined as the following.

Definition 13 (Term Query)**.** *A term query is represented by a bag of terms denoted Q_T. The keyword part q_{text} of Q_T is composed of top-n terms ranked by term weighting, i.e., TF-IDF, where the terms are extracted from d^q.*

Q_T is considered a bag of *terms* important to both d^q (locally) and the whole collection (globally). We also ignore the weight of terms (i.e., TF-IDF), that is, all terms are equally weighted at retrieval time. In contrast to the previous query type, Q_T aims at retrieving predictions related to the topics of news article, which can be represented as *a set of informative terms*. As an example, the Q_T with $q_{text} = \langle \textit{poll}, \textit{bush}, \textit{war}, \textit{iraq} \rangle$ is extracted from the same document used in the Q_E example above. In this case, q_{text} will be matched with the TEXT field of the predictions.

The last type is called "combined query" and it is defined as follows.

Definition 14 (Combined Query)**.** *A combined denoted Q_C is a combination of an entity query and a term query. The keyword part of q_{text} of Q_C is composed of both top-m entities and top-n terms formed by concatenating Q_E and Q_T.*

In this work, we combine by using the "AND" operator. The idea is that a prediction should be related to both *key entities* and *important terms* extracted from d^q. An example of Q_C can be $q_{text} = \langle \textit{George Bush}, \textit{Iraq}, \textit{America} \rangle$ AND $\langle \textit{poll}, \textit{bush}, \textit{war}, \textit{iraq} \rangle$. We discuss how we select *top-m* and *top-n* in Section 9.6.

The last component of the query is the *temporal criteria* or q_{time} used for retrieving predictions on the time dimension; q_{time} is composed of two different time constraints. The first constraint is specified in order to retrieve only predictions that are *future* relative to the publication date of query's parent article, or $\textit{time}(d^q)$. The second constraint indicates that those predictions must belong to news articles published before $\textit{time}(d^q)$.

Definition 15 (Temporal Criteria)**.** *Temporal criteria q_{time} is composed of two different time constraints and both time constraints can be represented using a time interval as:*

(i) $(\mathit{time}(d^q), t_{max}]$

(ii) $[t_{min}, \mathit{time}(d^q))]$

where $(\mathit{time}(d^q), t_{max}] = [\mathit{time}(d^q), t_{max}] - \{\mathit{time}(d^q)\}$, and t_{max} and t_{min} are the maximum time in the future and the minimum time in the past respectively. At retrieval time, the first constraint will be matched with the field FUTURE_DATE of predictions, whereas the second constraint will be matched with the field PUB_DATE of predictions.

9.4 Features

In this section, we present features used for learning a ranking model for related news predictions. The model will be described in Section 9.5. We propose several classes of features to capture the similarity between a news article query q and a prediction p, i.e., term similarity, entity-based similarity, topic similarity, and temporal similarity. The detailed description of each class will be given next.

9.4.1 Term Similarity

Since a prediction is defined with multiple fields, we employ the fielded searching provided with Apache Lucene search engine. The first term similarity feature *retScore* is the default similarity scoring function of Lucene [5], which is a variation of the tf-idf weighting scheme. The feature *retScore* will be computed with respect to a search field f, which is different for each query type. Note that, f is equivalent to the field ENTITY for Q_E, and the field TEXT for Q_T. For Q_C, *retScore* will be computed separately for each sub-query, and combine them into a final score. *retScore* is given as follows.

$$\mathit{retScore}(q,p,f) = \mathit{coord}(q,p) \cdot \mathit{qnorm}(q) \cdot \sum_{w_i \in q} \mathit{tf}(w_i,p) \cdot \mathit{idf}(w_i)^2 \cdot \mathit{boost}(w_i) \cdot \mathit{norm}(w_i,p) \quad (9.1)$$

$$\mathit{tf}(w_i,p) = \sqrt{\mathit{freq}(w_i,p)} \quad (9.2)$$

$$\mathit{idf}(w_i) = 1 + \log \frac{N_P}{n_{w_i}+1} \quad (9.3)$$

$$\mathit{qnorm}(q) = \frac{1}{\sqrt{\mathit{boost}(q)^2 \cdot \sum_{w_i \in q} (\mathit{idf}(w_i) \cdot \mathit{boost}(w_i))^2}} \quad (9.4)$$

$$\mathit{norm}(w_i,p) = \mathit{boost}(p) \cdot \mathit{lenNorm}(f) \cdot \mathit{boost}(f) \quad (9.5)$$

where $\mathit{tf}(w_i,p)$ is term frequency, and $\mathit{freq}(w_i,p)$ is a raw frequency of w_i in the field f of p. $\mathit{idf}(w_i)$ is an inverse prediction frequency. N_P is the total number of predictions and n_{w_i} is the number of prediction containing w_i. $\mathit{boost}(w_i)$ is the Lucene's term boosting

parameter for w_i in q. *coord*(q, p) is a score factor based on how many of the query terms are found in the specified prediction. *qnorm*(q) is a normalizing factor used to make scores between queries comparable. Calculating at indexing time, *norm*(w_i, p) encapsulates a document boost, a field boost, and a length factors or the normalization value for a field f given the total number of terms contained in f. Finally, *retScore*(q, p) must be normalized to have a value between 0 and 1 by dividing by $\max_{\mathcal{P}_q}$ *retScore*(q, p) where $\mathcal{P}_q$ is a set of all retrieved predictions.

A disadvantage of *retScore* is that it will not retrieve any predictions that do not match the query terms. This issue is exacerbated in sentence retrieval by the fact that we have to retrieve short fragments of text which might refer to the query terms using anaphora or other linguistic phenomena. One technique to overcome this problem is to use query expansion/reformulation using synonyms or different words with very similar meanings. It has also been shown that extending a sentence structure by its surrounding *context* sentences and weighting them using a field aware ranking function like bm25f consistently improves sentence retrieval [15]. Therefore, rather than reformulating a query, we will retrieve a prediction by looking at the CONTEXT and TITLE fields, in addition to the TEXT field. Thus, even if the TEXT field does not match exactly with a query term, p can receive a score if either the CONTEXT or TITLE field match the query term.

In our case, instead of weighting differently keyword matches in the title or body of a Web page, we assign a different importance to matches in the sentence itself or its context. The second term similarity feature *bm25f* can be computed as follows.

$$\textit{bm25f}(q, p, F) = \sum_{w_i \in q} \frac{\textit{weight}(w_i, p)}{k_1 + \textit{weight}(w_i, p)} \cdot \textit{idf}(w_i) \tag{9.6}$$

$$\textit{weight}(w_i, p) = \sum_{f \in F} \frac{\textit{freq}(w_i, f) \cdot \textit{boost}(f)}{(1 - b_f) + b_f \cdot \frac{l_f}{avl_f}} \tag{9.7}$$

$$\textit{idf}(w_i) = \log \frac{N_P - n_{w_i} + 0.5}{n_{w_i} + 0.5} \tag{9.8}$$

where l_f is the field length, avl_f is the average length for a field f, b_f is a constant related to the field length, k_1 is a free parameter and *boost*(f) is the boost factor applied to a field f. N_P is the total number of predictions and n_{w_i} is the number of prediction containing w_i, and $F = \{\text{TEXT}, \text{CONTEXT}, \text{TITLE}\}$. We discuss parameter settings in Section 9.6.1.

9.4.2 Entity-based Similarity

As mentioned earlier, term matching prevents us to retrieve predictions semantically similar to a query but using different terms. For example, a query "George W. Bush" will not be able to match the following prediction because it does not contain exactly the query term, but "Mr. Bush".

> *Mr. Bush's plan calls for an immediate start to all personal income tax cuts now scheduled for 2004 and 2006, with the greatest gains going to the most well-off Americans.*

Thus, we exploit a named entity recognition method with semantically tagging [22] used during the document annotation process (cf. Section 9.3.1).

This feature class is aimed at measuring the similarity between q and p by measuring the similarity of the entities they each contain. Note that, this class is only applicable for a query consisting of entities, that is, Q_E and Q_C, and it is ignored for Q_T. The first feature *entitySim* compares a string similarity between the entities of q and p_{entity} using the Jaccard coefficient, which can be computed as follows.

$$entitySim(q, p) = \frac{|q \cap p_{entity}|}{|q \cup p_{entity}|} \tag{9.9}$$

where p_{entity} is a set of entities, $|q \cap p_{entity}|$ and $|q \cup p_{entity}|$ are the size of intersection and union of entities of q and p.

Thus, the higher the overlap between the entities of a prediction and the query, the higher the prediction will be ranked for the query. We also want to rank predictions by using features that are commonly employed in an *entity ranking* task. For example, an entity is relevant if it appears in the title of a document, or it always occurs as a subject of sentence. We will employ *entity ranking* features by assuming that the more relevant entities a prediction contains, the more relevant it is. The entity-based features will be extracted and computed relative to the parent document of a prediction (d^p) or on the prediction itself (p).

Features extracted from documents are *title*, *titleSim*, *senPos*, *senLen*, *cntSenSubj*, *cntEvent*, *cntFuture*, *cntEventSubj*, *cntFutureSubj*, *timeDistEvent*, *timeDistFuture* and *tagSim*. Features extracted from predictions are *isSubj* and *timeDist*. The value of all features is normalized to range from 0 to 1, unless otherwise stated. First, the feature *title* indicates whether an entity e is in the title of d^p.

$$title(e, d^p) = isInTitle(e, d^p) \tag{9.10}$$

A value is 1 if e appears in the title of d^p, or 0 if otherwise. *titleSim* is a string similarity between e and the title.

$$titleSim(e, d^p) = \frac{|e \cap title(d^p)|}{|e \cup title(d^p)|} \tag{9.11}$$

senPos gives the position of the 1^{st} sentence where e occurs in d^p.

$$senPos(e, d^p) = \frac{len(d^p) - pos(firstSen(e))}{len(d^p)} \tag{9.12}$$

where $len(d^p)$ gives the length of d^p in words. $pos(s_y)$ is the position of a sentence s_y in d^p. *senLen* gives the length of the first sentence of d that contains e.

$$senLen(e, d^p) = \frac{len(firstSen(e))}{\max_{s_y \in d^p} len(s_y)} \tag{9.13}$$

cntSenSubj is the number of sentences where e is a subject. We run a dependency parser over the sentences in order to determine whether an entity is a subject of not.

$$\textit{cntSenSubj}(e, d^p) = \frac{1}{|\mathcal{S}_e|} \sum_{s_y \in \mathcal{S}_e} \textit{isSubj}(e, s_y) \tag{9.14}$$

where $\mathcal{S}_e$ is a set of all sentences of e in d^p. *isSubj*(e, s_y) is 1 if e is a subject of s_y. *cntEvent* is the number of event sentences (or sentences annotated with dates) of e.

$$\textit{cntEvent}(e, d^p) = \frac{1}{|\mathcal{E}_d^p|} \sum_{s_z \in \mathcal{E}_d^p} \sum_{s_y \in \mathcal{S}_e} \textit{isEqual}(s_z, s_y) \tag{9.15}$$

where $\mathcal{E}_d^p$ is a set of all event sentences in d^p. *isEqual*(s_z, s_y) returns 1 if s_z equals to s_y. *cntFuture* is the number of sentences with a mention of a future date. *cntEventSubj* is the number of event sentences that e is a subject.

$$\textit{cntEventSubj}(e, d^p) = \frac{1}{|\mathcal{E}_d^p|} \sum_{s_z \in \mathcal{E}_d^p} \textit{isSubj}(e, s_z) \tag{9.16}$$

Similarly, *cntFutureSubj* is the number of future sentences that e is a subject. *timeDistEvent* is a measure of the distance between e and all dates in d^p.

$$\textit{timeDistEvent}(e, d^p) = \frac{1}{|\mathcal{E}_e|} \sum_{s_z \in \mathcal{E}_e} \textit{avg}(\textit{normist}(e, s_z)) \tag{9.17}$$

$$\textit{normDist}(e, s_z) = \frac{1}{|\mathcal{T}_{s_z}|} \sum_{t_k \in \mathcal{T}_{s_z}} \frac{\textit{maxDist}(s_z) - \textit{dist}(e, t_k)}{\textit{maxDist}(s_z)} \tag{9.18}$$

$$\textit{dist}(w_i, w_j) = |\textit{pos}(w_i) - \textit{pos}(w_i)| - 1 \tag{9.19}$$

where $\mathcal{E}_e$ is a set of all event sentences of e, and $\mathcal{T}_{s_z}$ is a set of all temporal expressions in s_z. *dist*(w_i, w_j) is a distance in words between terms w_i and w_j. *maxDist*(s_z) is a maximum distance between terms in s_z. *timeDistFuture*(e, d^p) is a distance of e and all future dates in d^p computed similarly to *timeDistEvent*. *tagSim* is a string similarity between e and an entity tagged in d^p.

$$\textit{tagSim}(e, d^p) = \max_{e_n \in \mathcal{N}_d^p} \frac{|e \cap e_n|}{|e \cup e_n|} \tag{9.20}$$

where $\mathcal{N}_d^p$ is a set of all entities tagged in d^p. *tagSim* is only applicable for a collection provided with manually assigned tags (e.g., the New York Times Annotated Corpus).

isSubj(e, p) is 1 if e is a subject with respect to a prediction p, and *timeDist*(e, p) is a distance of e and all future dates in p computed similarly to *timeDistEvent*. All features in this class are parameter-free.

9.4.3 Topic Similarity

This class of features is aimed to compare the similarity between q and p on a higher level by representing them using topics. Examples of topics are "health care reform", "financial crisis", and "global warming". Several works [16, 128] have proposed to model a document with a low dimensionality, or to use topics rather than terms. We will use latent Dirichlet allocation (LDA) [16] to model a set of topics. LDA is based on a generative probabilistic model that models documents as mixtures over an underlying set of topic distributions. In general, topic modeling consists of two main steps. The first step is to learn topic models from training data. The output from this step is the probabilistic distribution over each topic. LDA requires the parameter N_z or the number of topics to be specified. After a model is trained, the next step is to infer topics from the learned topic model outputting a topic distribution for the prediction.

Wei and Croft [128] incorporated topic modeling for ad-hoc retrieval, and showed that linearly combining LDA with the query likelihood model outperformed non-topic models like the unigram model. We incorporate LDA into the retrieval process differently from Wei and Croft in two ways. First, instead of combining LDA scores with the original retrieval score, we represent q and p as vectors of topic distributions and compute the topic-based similarity using a cosine similarity between two vectors. Second, we explicitly take the time dimension into modeling topics because topics distributions can evolve over time. Intuitively, topics keep changing over time according to different trends.

We apply topic modeling to future retrieval in three main steps:

1. learning a topic model
2. inferring topic models
3. measuring topic similarity

Learning a topic model. We take into account the time dimension for learning topic models. As shown in Figure 9.2, we create training data by partitioning the document collection D_N into sub-collections (or document snapshots) with respect to time. In other words, we group documents by year of publication, and randomly select documents as training data, called a training data snapshot D_{train,t_k} at time t_k. Note that, we can also use more sophisticated approaches for modeling topics over time as presented in [125]. However, we will leave this study for future work.

Topic model inference. Using learned models from the previous step, we determine the topics for q and p from their contents. This process is called *topic inference*, which represents a query and a prediction by a distribution of topics (probabilities). For example, given a topic model ϕ, a prediction p can be represented as $p_\phi = p(z_1), \ldots, p(z_n)$, where $p(z)$ gives a probability of a topic z obtained from ϕ. Because our topic models are learned from different time periods, a question is which model snapshot we use for inference. Note that, q and p must be inferred from the same model snapshot in order to be comparable. We select a topic model for inferring in two ways. First, we select a topic model from a time snapshot $time(d^q)$ which corresponds to the publication date of the news article parent of q. Second, a topic model is selected from a time snapshot

t which corresponds to the publication date of the news article making prediction p, or the *time*(d^p). Moreover, a prediction p will be inferred in three different ways depending on the contents used: 1) only text p_{txt}, 2) both text p_{txt} and context p_{ctx}, and 3) the parent document d^p. For a query q, the contents of its parent document d^q will be used for inference.

In addition to using all N_z topics for inference, we will also select only top-k topics ranked by the importance. The idea is that measuring the topic similarity using too many topics may not be as accurate as using only the most important topics. We use *coverage* and *variation* proposed in [112] for ranking topics. A topic coverage $\mu(z)$ assumes that topics that cover a significant portion of the corpus content are more important than those covering little content, while a topic variation $\sigma(z)$ considers topics that appear in all the documents to be too generic to be interesting, although they have significant content coverage. $\mu(z)$ and $\sigma(z)$ are computed using a mean and a standard deviation over topic distributions, and the final score for ranking topic is a multiply of $\mu(z)$ and $\sigma(z)$. The calculation $\mu(z)$ and $\sigma(z)$ for a topic z at time t_k is given as:

$$\mu(z) = \frac{1}{\sum_{i=1}^{N_D} \textit{len}(d_i)} \sum_{i=1}^{N_D} \textit{len}(d_i) \cdot p_i(z) \tag{9.21}$$

$$\sigma(z) = \sqrt{\frac{1}{\sum_{i=1}^{N_D} \textit{len}(d_i)} \sum_{i=1}^{N_D} \textit{len}(d_i) \cdot (p_i(z) - \mu(z))^2} \tag{9.22}$$

where N_D is the number of documents in a training set at time t_k, or $|D_{train,t_k}|$. $p_i(z)$ gives a probability of a topic z in a document d_i and *len*(d_i) is the document length of d_i. A final score for ranking a topic z can be computed as:

$$\textit{rank}(z) = \mu(z)^{\lambda_1} \cdot \sigma(z)^{\lambda_2} \tag{9.23}$$

where the parameters λ_1 and λ_2 indicate the importance of $\mu(z)$ and $\sigma(z)$. If $\lambda_1 = 1$ and $\lambda_2 = 0$, the ranking is determined purely by topic coverage. On the contrary, if $\lambda_1 = 0$ and $\lambda_2 = 1$, the ranking emphasizes topic variance.

Measuring topic similarity. Given a topic model ϕ, the topic similarity can be calculated using a cosine similarity between a topic distribution of query q_ϕ and a topic distribution of prediction p_ϕ as follows.

$$\begin{aligned} \textit{topicSim}(q, p) &= \frac{q_\phi \cdot p_\phi}{||q_\phi|| \cdot ||p_\phi||} \\ &= \frac{\sum_{z \in Z} q_{\phi_z} \cdot p_{\phi_z}}{\sqrt{\sum_{z \in Z} q_{\phi_z}^2} \cdot \sqrt{\sum_{z \in Z} p_{\phi_z}^2}} \end{aligned} \tag{9.24}$$

We denote a topical feature using $LDA_{i,j,k}$, where i is one of the two different methods for selecting model snapshot: $i = 1$ for selecting a topic model from a time snapshot *time*(d^q), and $i = 2$ for selecting from a time snapshot *time*(d^p); j is one of the three

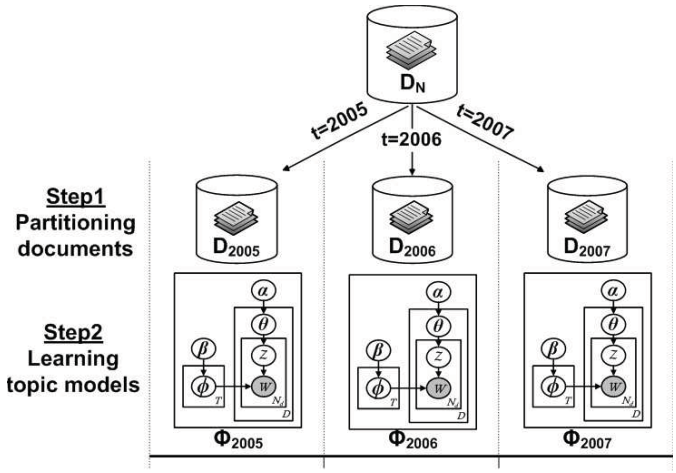


Figure 9.2: LDA topic snapshots based on time.

different ways of using the contents for inference: p_{txt}, p_{ctx}, or d^p. Finally, k refers to whether we use *all* of only top-k of topics for inference. Thus, this results in 12 (=3*2*2) LDA-based features in total.

9.4.4 Temporal Similarity

As mentioned earlier, we explicitly exploit temporal expressions in ranking. To measure the temporal similarity between a query and a prediction, we employ two features proposed in previous work: *TSU* (cf. Chapter 4) and *FS* [54].

We will represent our model of time using a time interval $[b, e]$ having a begin point b and the end point e. The actual value of any time point, e.g., b or e in $[b, e]$, is represented using the time model of Berberich et al. [10] presented in Section 2.2.2.

The first feature *TSU* is defined as the probability of generating the time of query q_{time} from the document creation date $time(d)$. *TSU* can be computed as follows.

$$TSU = DecayRate^{\lambda \cdot \frac{|q_{time} - time(d)|}{\mu}} \tag{9.25}$$

where $DecayRate$ and λ are constants, $0 < DecayRate < 1$ and $\lambda > 0$. μ is a unit of time distance. Intuitively, the probability obtained from this function decreases proportional to the distance between q_{time} and $time(d)$, that is, a document with its creation date closer to q_{time} will receive a higher probability than a document with its creation date farther from q_{time}.

We apply *TSU* for measuring the temporal similarity between q and p based on two assumptions. First, we assume that p is more likely to be relevant if its parent time $time(d^p)$ is closer to the time of query article $time(d^q)$. Our first temporal feature, denoted TSU_1,

will be calculated similarly to Equation 9.25 resulting the following function.

$$TSU_1(q,p) = DecayRate^{\lambda \cdot \frac{|time(d^q) - time(d^p)|}{\mu}} \tag{9.26}$$

The second assumption, denoted TSU_2, is that a prediction is more likely to be relevant if its future dates p_{future} are closer to the publication date of query article $time(d^q)$. If there are more than one future dates associated to p, a final score will be averaged over scores of all future dates p_{future}. The temporal distance of TSU_2 of q and p is defined as follows.

$$TSU_2(q,p) = \frac{1}{N_f} \sum_{t_f \in p_{future}} DecayRate^{\lambda \cdot \frac{|time(d^q) - t_f|}{\mu}} \tag{9.27}$$

where t_f is a future date in p_{future} and N_f is the number of all future dates.

In addition to TSU_1 and TSU_2, we can measure the temporal similarity between q and p using a fuzzy membership function, which is originally proposed by Kalczynski and Chou [54].

We adapt the original fuzzy set function in [54] by using its parent time $time(d^p)$ and the time of query article $time(d^q)$. We denote this feature as FS_1, and it can be computed as follows.

$$FS_1(q,p) = \begin{cases} 0 & \text{if } time(d^p) < \alpha_1 \vee time(d^p) > time(d^q), \\ f_1(time(d^p)) & \text{if } time(d^p) \geq \alpha_1 \wedge time(d^p) < time(d^q), \\ 1 & \text{if } time(d^p) = time(d^q). \end{cases} \tag{9.28}$$

$$f_1(time(d^p)) = \begin{cases} \left(\frac{time(d^p) - \alpha_1}{time(d^q) - \alpha_1}\right)^n & \text{if } time(d^p) \neq time(d^q), \\ 1 & \text{if } time(d^p) = time(d^q). \end{cases} \tag{9.29}$$

We define the second temporal feature based on a fuzzy set by using the prediction's future dates p_{future} and the publication date of query article $time(d^q)$. Similarly, if a prediction p has more than one future date, a final score will be averaged over scores of all dates p_{future}. The second temporal feature FS_2 is defined as follows.

$$FS_2(q,p) = \frac{1}{N_f} \sum_{t_f \in p_{future}} \begin{cases} 0 & \text{if } t_f < time(d^q) \vee t_f > \alpha_2, \\ 1 & \text{if } t_f = time(d^q), \\ f_2(t_f) & \text{if } t_f > time(d^q) \wedge t_f \leq \alpha_2. \end{cases} \tag{9.30}$$

$$f_2(t_f) = \begin{cases} \left(\frac{\alpha_2 - t_f}{\alpha_2 - time(d^q)}\right)^m & \text{if } t_f \neq time(d^q), \\ 1 & \text{if } t_f = time(d^q). \end{cases} \tag{9.31}$$

where N_f is the number of all future dates in p_{future}, and t_f is a future date, i.e., $t_f \in p_{future}$. n and m are constants. α_1 and α_2 are the minimum and maximum time of reference with respect to q_{time}. α_1 is calculated by subtracting the time offset s_{min} from from q_{time} , and α_2 is calculated by adding the offset s_{max} to q_{time}.

9.5 Ranking Model

Given a query q, we will rank a prediction p using a ranking model obtained by training over a set of labeled query/prediction pairs using a learning algorithm. An unseen query/prediction pair (q, p) will be ranked according to a weighted sum of feature scores:

$$score(q, p) = \sum_{i=1}^{N} w_i \times x_i \tag{9.32}$$

where x_i are the different features extracted from p and q, N is the number of features, and w_i are the weighting coefficients. The goal of the algorithm is to learn the weights w_i using a training set of queries and predictions, in order to minimize a given loss function. Learning to rank algorithms can be categorized into three approaches: pointwise, pairwise, and listwise approaches [77]. The pointwise approach assumes that retrieved documents are independent, so it predicts a relevance judgment for each document and ignores the positions of documents in a ranked list. The pairwise approach considers a pair of documents, and relevance prediction is given as the relative order between them (i.e., pairwise preference). The listwise approach considers a whole set of retrieved documents, and predicts the relevance degrees among documents. For a more detailed description of each approach, please refer to [77].

We employ the listwise learning algorithm SVM^{MAP} [136]. The algorithm trains a classifier using support vector machines (SVM), and it determines the order of retrieved documents in order to directly optimize Mean Average Precision (MAP). In addition, we also experimented with other learned ranking algorithms: RankSVM [52], SGD-SVM [142], PegasosSVM [110], and PA-Perceptron [23]. However, these algorithms do not perform as well as SVM^{MAP} in our experiments. Thus, we will only discuss the results obtained from SVM^{MAP} in the next section.

9.6 Evaluation

In this section, we evaluate the retrieval effectiveness of our proposed ranking model using three different query formats. We will first describe the experimental settings followed by an explanation of the results and a detailed discussion.

9.6.1 Setting

Document collection. We used the New York Times Annotated Corpus for our document collection. In order to extract predictions and features, a series of language processing tools, including OpenNLP [97] (for tokenization, sentence splitting and part-of-speech tagging, and shallow parsing), the SuperSense tagger [114] (for named entity recognition) and TARSQI Toolkit [119] (for extracting temporal expressions from documents). Given the importance of time to our system, we note that the temporal expression extraction of TARSQI has a reported performance of 0.81 F1 on the Time Expression Recognition and Normalization task [121].

Table 9.2: Examples of future-related topics.

POLITICS	ENVIRONMENT	SPACE
president election	global warming	Mars
Iraq war	energy efficiency	Moon
SCIENCE	**PHYSICS**	**HEALTH**
earthquake	particle Physics	bird flue
tsunami	Big Bang	influenza
BUSINESS	**SPORT**	**TECHNOLOGY**
subprime	Olympics	Internet
financial crisis	World cup	search engine

We employed the Apache Lucene search engine for both indexing and retrieving predictions. The statistics of extracted data are as follows. There are 44,335,519 sentences and 548,491 are predictions. There are 939,455 future dates, and an average future date per prediction is 1.7 and the standard deviation is 0.92. Among 1.8 million documents, more than 25% of all documents contain at least one prediction (i.e., a reference to the future). In order to determine this percentage over a broader range of news sources, we performed the same analysis on 2.5 million documents from over 100 news sources from Yahoo! News for the one year period from July 2009 to July 2010 and found over 32% of the documents contained at least one prediction.

Future-related queries. There is no gold standard available to evaluate the task of ranking related news prediction. We manually selected 42 query news articles from the New York Times that cover the future-related topics shown in Table 9.2. The actual queries (Q_E, Q_T and Q_C) used for retrieving predictions are extracted from these news articles.

Relevance assessments. Human assessors were asked to evaluate query/prediction pairs (e.g., relevant or non-relevant) using 5 levels of relevance: 4 for *excellent* (very relevant prediction), 3 for *good* (relevant prediction), 2 for *fair* (related prediction), 1 for *bad* (non-relevant prediction), and 0 for *non prediction* (incorrect tagged date). The last option was presented because there are predictions incorrectly annotated with time (this is an error produced by the annotation tools). More precisely, an assessor was asked to give a relevance score $Grade(q, p, t)$ where (q, p, t) is a triple of a query q, a prediction p, and a future date t in p. Consider the following prediction about the topic "global warming" and the publication date of the news article is 2007/02/21:

> *Formal ratification of the pact – which commits the union to reduce emissions of "greenhouse gases" by 8 percent of 1990 levels during the five-year period from* **2008** *through* **2012** *– now goes to the European Council of heads of state and government, which could act as early as this month at the union summit in Barcelona.*

The prediction contains two future dates (as highlighted in bold). Hence, an assessor has to give judges to two triples corresponding to q, p and *both* future dates. A triple

(q, p, t) is considered **relevant** if $Grade(q, p, t) \geq 3$, and it is considered non-relevant if $1 \leq Grade(q, p, t) \leq 2$. Relevance level 0 is not included in the evaluation[1]. These judgments are normalized by a query/prediction pair (q, p) since we are interested in presenting a prediction for all future dates, regardless of their number. That is, a query/prediction pair (q, p) is relevant if and only if there is at least one relevant triple (q, p, t), and a prediction is **non-relevant** if all triples are non-relevant. Our assumption is that predictions extracted from more recent documents are more relevant.

In total, assessors judged 52 queries and for each one of them we retrieved up to 100 sentences that contained predictions. On average 94 sentences with future mentions were retrieved, with an average of 1.2 future dates per prediction. Finally, assessors evaluated 4,888 query/prediction pairs (approximately 6,032 of triples).

Our machine learning ranking models operate in a supervised manner, and as such, they need training data for learning. We created training data using cross validation by randomly partitioned query articles into N_F folds. We used $N_F - 1$ query/prediction from other folds for training a ranking mode and the remaining fold for testing. We removed queries with zero relevant results, and we obtained $N_F = 3, 4, 5$ for Q_E, Q_C, Q_T respectively.

Parameter setting. The boost factors are set on independent experiments: $boost(\text{TEXT}) = 5.0$, $boost(\text{CONTEXT}) = 1.0$, and $boost(\text{TITLE}) = 2.0$. We use the recommended values [104] for the constants $b = 0.75$ for all fields, and $k_1 = 1.2$. For LDA-based features, we trained a yearly model snapshot by selecting 4% of all documents in each year. For each document, we filtered out terms occurring in less than 15 documents and the 100 most common terms. We learn a topic model for each document snapshot by employing Stanford Topic Modeling Toolbox [122], and the number of topics for training LDA N_z is fixed to 500 and the number of topics for inference k is 200. A learning algorithm we use is the collapsed variational Bayes approximation to the LDA objective (CVB0LDA) [6]. All other parameters are default values of the topic modeling. Using CVB0LDA required high CPU and memory, but needed fewer iterations and had faster convergence rates than a collapsed Gibbs sampler [39], which requires less memory during training.

For both TSU_1 and TSU_2, $DecayRate = 0.5$, $\lambda = 0.5$ and $\mu = 2y$ are used where y the number of years. For both FS_1 and FS_2, $n = 2$, $m = 2$, $s_{min} = 4y$ and $s_{max} = 2y$ are used. So, $\alpha_1 = time(d^q) - 4y$ and $\alpha_2 = time(d^q) + 2y$.

Methods for comparison. We experiment with the three different ways of constructing the query Q_E, Q_T, and Q_C. The baseline for retrieval is Lucene's default ranking function and our queries incorporate two time constraints as explained in Section 9.3.4. We re-rank the baseline results using SVM^{MAP} yielding *Re-*Q_E, *Re-*Q_T and *Re-*Q_C. For the application of ranking related news predictions, we prefer top-precision retrieval performance metrics over recall-based metrics: a user will be typically interested in a few top predictions even though there are many predictions retrieved. Consequently, we envision a user interface that contains little space for displaying related predictions. Thus, we will measure the retrieval effectiveness by the precision at 1, 3 and 10 (P@1, P@3, and P@10 respectively), Mean Reciprocal Rank (MRR), and Mean Average Precision (MAP). We

[1] We are interested in assessing the performance of the ranking algorithm and not the annotation tools. However, we note the overall system will be impacted by the annotation errors.

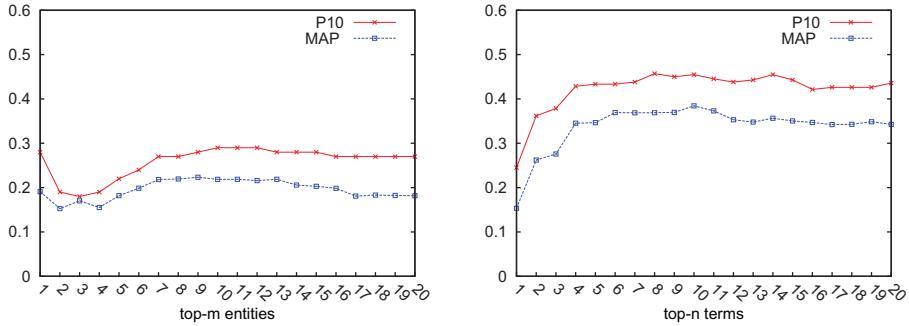


Figure 9.3: P@10 and MAP performance of Q_E (left) when varying top-m entities, and Q_C (right) when varying top-n terms.

report the average performance over N_F folds to measure the overall performance, for each query type.

9.6.2 Results

The three types of queries (Q_E,Q_T, and Q_C) are composed of either *top-m* entities or *top-n* terms, or both. We first establish which are good m and n values for each one of the types. Instead of varying m and n in re-ranking, we select the m and n that give a reasonable improvement in a hold-out set (where we randomly divided queries into two folds). Therefore, we will use only one fixed version of m and n for the rest of our experiments. We select the values of m and n by performing a preliminary analysis as follows. First, by looking at P@10 and MAP, we select the value of m that yields the best performance using only Q_E to retrieve predictions for each varying m. As shown in Figure 9.3 (left), $9 \leq m \leq 12$ give almost no difference in terms of P@10. In spite of that, we choose the number of entities $m = 11$ because it is slightly better than the other values. Next, we find the optimal value of n by observing the performance of Q_C when m is fixed to 11 and the value of n is varied. As depicted in Figure 9.3(right), there is very slight difference in P@10 for $9 \leq n \leq 11$; We choose the number of terms $n = 10$ because it obtains the best in MAP among them.

The retrieval effectiveness of simple methods and their corresponding re-ranking methods are displayed in Table 9.3. These results are averaged over queries retrieving at least one relevant prediction. In general, Q_T gains the highest effectiveness in all measurements followed by Q_C and Q_E and the feature-based re-ranking approach improves the effectiveness for all query types. In addition, *Re-*Q_C has the highest effectiveness over other re-ranking methods for P@1 and P@3, while *Re-*Q_T gains the highest effectiveness for the rest of all metrics.

Q_E and Q_C pose a problem in not retrieving any relevant result of our judged pool among the first 100 for a large number of queries, which makes it impossible for the machine learning model to improve the ranking. However, we still want to compare the

Table 9.3: Effectiveness of each method when using all queries; †,∓ indicates statistical improvement over the corresponding simple methods using t-test with significant at $p < 0.1$,$p < 0.05$ respectively.

Method	P@1	P@3	P@10	MRR	MAP
Q_E	0.300	0.333	0.290	0.473	0.219
Q_T	0.643	0.579	0.455	0.760	0.385
Q_C	0.500	0.561	0.427	0.656	0.231
Re-Q_E	0.500	0.499	0.360	0.629	0.266
Re-Q_T	0.738†	0.619	0.462	0.831†	0.387
Re-Q_C	0.773∓	0.682∓	0.455	0.841∓	0.271

performance between the different variations of the query (Q_E, Q_C, Q_T). Therefore, we use a subset of queries that contained *at least* one relevant result among all the different methods. The results are shown in Table 9.4 where we compare all other methods against Q_E because we have observed that Q_E performs worst among them. As seen from the results of each re-ranking method, our proposed features improve the effectiveness for all corresponding simple methods. In particular, the re-ranking method *Re-*Q_C outperforms the simple method Q_E significantly. However, *Re-*Q_E did not provide a significant improvement over Q_E. The results show that, for the same set of queries, using entities alone are limited while terms alone are able to retrieve most of relevant predictions.

Interestingly, when looking at the same sub-set of queries with relevant predictions, the re-ranking approach *Re-*Q_C outperforms every other method, even if the plain retrieval Q_T is superior to Q_C. This is an indicator that entity-based features are able to produce higher quality results but only for a certain type of topics. We performed an error analysis to determine why Q_E is unable to retrieve relevant predictions. In general, Q_E fails for a topic that cannot be represented using only people, locations, or organizations. For example, for the topic about "the Europeans agreement of gas emissions", the top-5 Q_E is $\langle$*European Union*, *Brussels*, *Finland*, *Germany*, *Hungary*$\rangle$ and the top-5 Q_T is $\langle$*european*, *emission*, *target*, *climate*, *brussels*$\rangle$. In this case, Q_E is unable to represent the *key* terms "emission" and "climate", and thus fails to retrieve many relevant predictions that match those terms.

Similarly, for the query topic about "Clinton health care reform", Q_E is represented using the named entity *Clinton* (the terms "health care" and "reform" are not annotated as entities). When matching, all predictions containing the entity *Clinton* are matched which will return many documents that are not related to "health care" and "reform".

9.6.3 Feature Analysis

We analyzed feature weights obtained from the learning algorithm SVM^{MAP} in order to understand better what is the importance of the different features,. Note that, in order to compare the weights among different queries, we performed normalization by diving with the maximum value of all weights for each query. Column w_i in Table 9.5 displays

Table 9.4: Effectiveness of each method when using a subset of queries; †,∓,⋆ indicates statistical improvement over the method Q_E using t-test with significant at $p < 0.1$, $p < 0.05$, $p < 0.01$ respectively.

Method	P@1	P@3	P@10	MRR	MAP
Q_E	0.300	0.333	0.290	0.473	0.219
Q_T	0.500	0.533	0.430	0.638	0.219
Q_C	0.600†	0.533†	0.360	0.727†	0.163
Re-Q_E	0.500	0.499	0.360	0.629	0.266
Re-Q_T	0.700	0.600	0.410	0.762	0.236
Re-Q_C	1.000∓	0.714∓	0.443	1.000	0.303⋆

Table 9.5: Top-5 features with highest weights and lowest weights for each query type.

Q_E		Q_T		Q_C	
Feature	W_i	Feature	W_i	Feature	W_i
tagSim	1.00	*bm25f*	1.00	$LDA_{1,parent,k}$	1.00
FS_1	0.97	*retScore*	0.60	*retScore*	0.99
TSU_2	0.88	$LDA_{1,parent,k}$	0.55	$LDA_{1,parent,all}$	0.96
$LDA_{1,txt,k}$	0.87	$LDA_{2,parent,k}$	0.51	*bm25f*	0.93
$LDA_{1,txt,all}$	0.82	$LDA_{1,parent,all}$	0.49	*isSubj*	0.87
cntSenSubj	0.01	*timeDistEvent*	-0.03	*cntEventSen*	-0.02
cntEventSubj	0.01	*timeDistFuture*	-0.11	*querySim*	-0.05
isInTitle	0.00	*cntEventSen*	-0.12	*cntFutureSen*	-0.10
cntEventSen	0.00	*cntFutureSen*	-0.12	*timeDistFuture*	-0.14
querySim	-0.01	*senLen*	-0.16	*senLen*	-0.18

the top-5 features with highest and lowest weights for each query type.

At least two topic-based features of all query types are in the top-5 features with highest weight, and therefore topic-based features play an important role in the re-ranking model. Although *retScore* and *bm25f* measure the similarity on a term level, they help to re-rank predictions when incorporated into the machine learning model. as seen in the top-5 features for Q_T and Q_C. The feature that received the highest importance value for the Q_E type is *tagSim*, which measures the similarity between entities in a prediction and manually tagged entities. This indicates that tagged entities in a query document can precisely represent user information needs. The temporal features FS_1 and FS_1 also play an important role for Q_E.

Features in top-5 features with lowest weights are those from the entity-based class. Recall that these features are extracted in order to measure the importance of entities annotated in a prediction with respect to their respective parent documents. However, the results show that these features are not good enough for discriminating between relevant and non-relevant predictions.

In order to observe the performance of different classes of feature, we conducted two additional experiments for all query types. First, we trained a ranking model with SVM^{MAP} using only *retScore* and selected a class of features at each time to observe how the selected class contributes to the ranking model. For each query type, a baseline is the model trained using *retScore* only. We compare the effectiveness using P@3 and MRR. Besides, we also show the percentage of improvement compared to the baseline. The results are depicted in Table 9.6. For Q_E, the classes *term*, *entity* and *prediction* improve the baseline slightly or not at all. *temporal* and *topic* better the effectiveness, but not significantly. For Q_T, adding the class *entity* decreases the effectiveness as well as all other classes show the improvement to a small degree. Nevertheless, *term* performs better than the baseline significantly in P@3. For Q_C, all feature classes do not improve the baseline significantly except *topic* that outperforms the baseline significantly up to 24% in MRR. In addition to single classes, combining all features yield the improvement significantly in both P@3 and MRR.

The second experiment is training a ranking model using training data that are consisted of *all* features, but except one class at each time to see how the ranking model is depended on that class. We also use P@3 and MRR to compare the effectiveness, and we report how much the effectiveness is decreased (%) compared to the baseline. For Q_E, all other classes result in decreasing the effectiveness when dropped, while the effectiveness is slightly increase when *temporal* is removed. However, we cannot conclude that *temporal* has an negative effect to Q_E, because adding *temporal* alone can improve the baseline. For Q_T, dropping each single feature class alters the effectiveness of the baseline slightly, whereas dropping a combination of all classes decreases the effectiveness up to %20 in MRR. Similarly, for Q_C, dropping each single feature class changes the effectiveness of the baseline to some degree. However, a combination of all classes decreases the effectiveness significantly in both P@3 and MRR when dropped.

9.7 Conclusions

In this chapter, we demonstrated that future related information is abundant in news stories and defined the task of *ranking related future predictions*. The main goal of this task is to improve user access to this information by selecting the predictions from a news archive that are most relevant to a given news article. We created an evaluation dataset with over 6000 relevance judgments and addressed this task using a learning to rank methodology incorporating four classes of features including term similarity, entity-based similarity, topic similarity, and temporal similarity that outperforms a strong baseline system. Finally, we performed an in-depth analysis of feature importance.

Table 9.6: Effectiveness of different classes of features, * indicates statistically improvement over the baselines using t-test with significant at $p < 0.05$.

Type	Method	Add				Drop			
		P@3	(%)	MRR	(%)	P@3	(%)	MRR	(%)
Q_E	baseline	.33	-	.47	-	.50	-	.63	-
	term	.37	*4*	.50	*3*	.43	*7*	.58	*5*
	entity	.33	*0*	.49	*2*	.43	*7*	.63	*0*
	prediction	.33	*0*	.52	*5*	.40	*10*	.58	*5*
	temporal	.40	*7*	.55	*8*	.53	*-3*	.72	*-9*
	topic	.43	*10*	.63	*16*	.40	*10*	.60	*3*
	all	.50	*13*	.63	*16*	.33	*17*	.47	*16*
Q_T	baseline	.58	-	.76	-	.62	-	.83	-
	term	.65*	*7*	.79	*3*	.63	*-1*	.83	*0*
	entity	.56	*-2*	.74	*-2*	.63	*-1*	.83	*0*
	prediction	.58	*0*	.78	*2*	.63	*-1*	.85	*-2*
	temporal	.60	*2*	.80	*4*	.63	*-1*	.84	*-1*
	topic	.61	*3*	.80	*4*	.60	*2*	.78	*5*
	all	.62	*4*	.83	*7*	.58	*4*	.63	*20*
Q_C	baseline	.56	-	.64	-	.68	-	.83	-
	term	.59	*3*	.67	*3*	.63	*5*	.86	*-3*
	entity	.56	*0*	.66	*2*	.67	*1*	.88	*-5*
	prediction	.60	*4*	.75	*11*	.67	*1*	.88	*-5*
	temporal	.60	*4*	.70	*6*	.65	*3*	.85	*-2*
	topic	.62	*6*	.88*	*24*	.60	*8*	.72	*11*
	all	.68*	*12*	.83*	*19*	.56*	*12*	.64*	*19*

Chapter 10

Conclusions

This thesis addresses research problems in searching temporal document collections. We have proposed different approaches to solving the addressed research questions. In summary, our contributions to this thesis are:

- We proposed different techniques for improving temporal language models used for determining the creation time of non-timestamped documents. The proposed approaches included different semantic-based preprocessing. In addition, we improved the quality of document dating by incorporating internal and external knowledge into the temporal language models. By conducting extensive experiments, we showed the evaluation of our proposed approach and the improvement over the baseline of our proposed approaches. Finally, we presented a system prototype for dating documents using the proposed extension approaches.

- We performed the first study on how to determine the temporal profiles of queries without temporal criteria provided, and we proposed techniques for determining the time of implicit temporal queries. We proposed an approach to re-ranking search results by incorporating the determined time of queries. By conducting extensive experiments, we evaluated our approaches for determining temporal profiles of queries, as well as of re-ranking search results using temporal profiles of queries.

- We formally modeled Wikipedia viewed as a temporal resource for classification of time-based synonyms. We proposed an approach to discovering time-based synonyms using Wikipedia and improving the time of synonyms. In addition, we proposed query expansion techniques that exploit time-based synonyms. We extensively evaluated our proposed approaches to extracting and improving time of synonyms, and query expansion using time-based synonyms. Finally, a news archive search prototype considering terminology changes over time was presented.

- We performed the first study and analysis of performance prediction methods for temporal queries. We proposed different time-based predictors and techniques for

improving query performance prediction by combining multiple predictors. Extensive experiments were conducted for evaluating single predictors and the combined methods.

- We conducted the first study of query classification using two time dimensions, and presented a novel taxonomy of queries based on two time dimensions was formally defined. We proposed an approach to automatically classifying a query into two main classes as well as retrieval models for both time dimension. In addition, an approach to predicting an appropriate ranking model for time-sensitive queries was presented. By performing extensive experiments, we evaluated our proposed query classification and time-aware ranking model prediction.

- We performed the first study and analysis of time-aware ranking methods. By conducting extensive experiments, we compared the performance of different time-aware ranking methods using the same dataset.

- We proposed the first study of the ranking related news predictions task. We proposed a learned ranking model incorporating four classes of features including term similarity, entity-based similarity, topic similarity, and temporal similarity. We evaluated our proposed approach using queries selected from real-world future trends and predictions extracted from the New York Times Annotated Corpus. Finally, we performed an in-depth analysis of feature selection to guide further research in the ranking related news predictions task.

In the rest of this chapter, we will outline our plans for future work, and discuss possible research topics beyond what have been addressed in the thesis.

Future Work

Our plans for future work are as follows. In order to overcome the limitation of temporal language models, there are several issues we intend to study as part of the future research on *document/query dating*. First, our word interpolation method is an interesting idea in improving the language model. However, not every word should be interpolated in the same manner, thus we could apply a weighting scheme to words and interpolate only significant words. In addition, we intend to further improve the dating task based on external knowledge from sources such as Wikipedia. Finally, the dating task is analogous to a classification problem where a document is classified into a time interval. A possible future work is to employ different classification methods in determining time of a document.

Future work on *terminology changes over time* include combining time-dependent synonyms and temporal language models in order to provide temporal search using a named entity without having to provide explicitly the time in the query. We will also integrate our approach for time-dependent synonym discovery with information extraction techniques that can find additional information in Wikipedia (for example names of presidents at particular points in time). Finally, we want to exploit the detected relationships

together with temporal dynamics of document contents [34] in order to improve relevance ranking.

Our planned future work on *query performance prediction* is multifold. First, we want to find new time-based prediction methods that are more sophisticated than those we have proposed. For example, we can also consider time uncertainty as an indicator for query performance. Second, we want to apply time-based post-retrieval prediction for temporal search. This is because post-retrieval prediction determines a ranked list of retrieved documents, so in general post-retrieval prediction performs better than pre-retrieval prediction. In addition, we plan to increase the number of temporal queries used for analysis and also apply different temporal ranking methods. The most important next step is when a query is predicted to perform poorly, we want to apply time-based query expansion in order to improve the overall effectiveness of a temporal search system.

Outlook

The research problems we have addressed are among several problems of searching temporal document collections. Beyond the scope of this thesis, there are still a lot of opportunities for further investigation. In the following, we outline two promising topics for future research: 1) mining user-generated contents, and 2) spatio-temporal ranking models.

Mining user-generated contents. In recent years, social network services, e.g., Twitter, blogs, and discussion forums, have gained increasing interests. The contents generated from social networks (called user-generated contents) have increased dramatically, and challenges when dealing with such data include real-time (stream) data, noisy texts (unedited language), and dynamic topics/events. An interesting research direction is to employ our time-aware approaches to *mining user-generated contents*. One example of possible applications is Online Reputation Management that is created to monitor social medias, such as Twitter, in order to detect contents or opinions about an products, people and organizations [4]. In addition to an opinion mining application, we can also mining news articles together with user-generated contents in order to predict future events, as been discussed in Chapter 9.

Spatio-temporal ranking models. This thesis only focuses on temporal queries, that is, those containing temporal information needs. In some cases, queries may contain not only time, but also geographic information. For example, a user may search for a particular event by issuing a query, which is composed of both a location and time. In recent work [113], Strötgen et al. proposed to use temporal and geographic expressions in documents' contents to measure the similarity between documents for a cross-language retrieval task. Similarly, it is interesting to exploit temporal and geographic information in ranking. Thus, an interesting research direction is to incorporate both temporal and geographic information into retrieval and ranking.

Bibliography

[1] O. Alonso and M. Gertz. Clustering of search results using temporal attributes. In *Proceedings of SIGIR'2006*, 2006.

[2] O. Alonso, M. Gertz, and R. A. Baeza-Yates. On the value of temporal information in information retrieval. *ACM SIGIR Forum*, 41(2):35–41, 2007.

[3] O. Alonso, M. Gertz, and R. A. Baeza-Yates. Clustering and exploring search results using timeline constructions. In *Proceedings of CIKM'2009*, 2009.

[4] E. Amigó, J. Artiles, J. Gonzalo, D. Spina, B. Liu, and A. Corujo. Weps3 evaluation campaign: Overview of the on-line reputation management task. In *Proceedings of CLEF (Notebook Papers/LABs/Workshops)*, 2010.

[5] Apache Lucene. `http://lucene.apache.org/`.

[6] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh. On smoothing and inference for topic models. In *Proceedings of UAI'2009*, 2009.

[7] R. A. Baeza-Yates. Searching the future. In *Proceedings of SIGIR workshop on mathematical/formal methods in information retrieval MF/IR 2005*, 2005.

[8] R. A. Baeza-Yates and B. A. Ribeiro-Neto. *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011.

[9] K. Balog, L. Azzopardi, and M. de Rijke. A language modeling framework for expert finding. *Inf. Process. Manage.*, 45(1):1–19, 2009.

[10] K. Berberich, S. Bedathur, O. Alonso, and G. Weikum. A language modeling approach for temporal information needs. In *Proceedings of ECIR'2010*, 2010.

[11] K. Berberich, S. Bedathur, T. Neumann, and G. Weikum. Fluxcapacitor: efficient time-travel text search. In *Proceedings of the 33rd VLDB*, 2007.

[12] K. Berberich, S. J. Bedathur, T. Neumann, and G. Weikum. A time machine for text search. In *Proceedings of SIGIR'2007*, 2007.

[13] K. Berberich, S. J. Bedathur, M. Sozio, and G. Weikum. Bridging the terminology gap in web archive search. In *Proceedings of WebDB'2009*, 2009.

[14] K. Berberich, M. Vazirgiannis, and G. Weikum. Time-aware authority ranking. *Internet Mathematics*, 2(3), 2005.

[15] R. Blanco and H. Zaragoza. Finding support sentences for entities. In *Proceeding of SIGIR'2010*, 2010.

[16] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3:993–1022, March 2003.

[17] C. Bøhn and K. Nørvåg. Extracting named entities and synonyms from wikipedia. In *Proceedings of AINA'2010*, 2010.

[18] R. C. Bunescu and M. Paşca. Using encyclopedic knowledge for named entity disambiguation. In *Proceedings of EACL'2006*, 2006.

[19] J. Canton. *The Extreme Future: The Top Trends That Will Reshape the World in the Next 20 Years*. Plume, 2007.

[20] D. Carmel and E. Yom-Tov. *Estimating the Query Difficulty for Information Retrieval*. Morgan & Claypool Publishers, 2010.

[21] Z. Chen, J. Ma, C. Cui, H. Rui, and S. Huang. Web page publication time detection and its application for page rank. In *Proceeding of SIGIR '2010*, 2010.

[22] M. Ciaramita and Y. Altun. Broad-coverage sense disambiguation and information extraction with a supersense sequence tagger. In *Proceedings of EMNLP'2006*, 2006.

[23] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *J. Mach. Learn. Res.*, 7:551–585, 2006.

[24] B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, USA, 1st edition, 2009.

[25] S. Cronen-Townsend, Y. Zhou, and W. B. Croft. Predicting query performance. In *Proceedings of SIGIR'2002*, 2002.

[26] N. Dai and B. D. Davison. Freshness matters: in flowers, food, and web authority. In *Proceeding of SIGIR '2010*, 2010.

[27] W. Dakka, L. Gravano, and P. G. Ipeirotis. Answering general time-sensitive queries. In *Proceeding of CIKM'2008*.

[28] V. de Boer, M. van Someren, and B. J. Wielinga. Extracting historical time periods from the web. *Journal of the American Society for Information Science and Technology*, 61:1888–1908, September 2010.

[29] F. de Jong, H. Rode, and D. Hiemstra. Temporal language models for the disclosure of historical text. In *Proceedings of AHC'2005 (History and Computing)*, 2005.

[30] G. Demartini, A. P. de Vries, T. Iofciu, and J. Zhu. *Overview of the INEX 2008 Entity Ranking Track*. 2009.

[31] F. Diaz and R. Jones. Using temporal profiles of queries for precision prediction. In *Proceedings of SIGIR'2004*, 2004.

[32] A. Dong, R. Zhang, P. Kolari, J. Bai, F. Diaz, Y. Chang, Z. Zheng, and H. Zha. Time is of the essence: improving recency ranking using twitter data. In *WWW '10: Proceedings of the 19th international conference on World wide web*, pages 331–340, New York, NY, USA, 2010. ACM.

[33] J. L. Elsas and S. T. Dumais. Leveraging temporal dynamics of document content in relevance ranking. In *Proceedings of WSDM'2010*, 2010.

[34] J. L. Elsas and S. T. Dumais. Leveraging temporal dynamics of document content in relevance ranking. In *Proceedings of WSDM'2010*, New York, NY, USA, 2010.

[35] A. Ernst-Gerlach and N. Fuhr. Generating search term variants for text collections with historic spellings. In *Proceedings of ECIR*. Springer, 2006.

[36] A. Ernst-Gerlach and N. Fuhr. Retrieval in text collections with historic spelling using linguistic and spelling variants. In *Proceedings of JCDL*, 2007.

[37] Google News Archive Search. `http://news.google.com/archivesearch/`.

[38] Google Zeitgeist. `http://www.google.com/press/zeitgeist.html/`.

[39] T. L. Griffiths. Finding scientific topics. *Proceedings of the National Academy of Science*, 101:5228–5235, Jan 2004.

[40] R. Gwadera and F. Crestani. Mining and ranking streams of news stories using cross-stream sequential patterns. In *Proceedings of CIKM'2009*, 2009.

[41] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction (Second Edition)*. Springer, 2009.

[42] C. Hauff, L. Azzopardi, and D. Hiemstra. The combination and evaluation of query performance prediction methods. In *Proceedings of ECIR'2009*, 2009.

[43] C. Hauff, L. Azzopardi, D. Hiemstra, and F. de Jong. Query performance prediction: Evaluation contrasted with effectiveness. In *Proceedings of ECIR'2010*, 2010.

[44] C. Hauff, D. Hiemstra, and F. de Jong. A survey of pre-retrieval query performance predictors. In *Proceedings of CIKM'2008*, 2008.

[45] C. Hauff, V. Murdock, and R. Baeza-Yates. Improved query difficulty prediction for the web. In *Proceeding of CIKM'2008*, 2008.

[46] B. He and I. Ounis. Inferring query performance using pre-retrieval predictors. In *Proceedings of SPIRE'2004*, 2004.

[47] J. He, M. Larson, and M. de Rijke. Using coherence-based measures to predict query difficulty. In *Proceedings of ECIR'2008*, 2008.

[48] J. Hu, L. Fang, Y. Cao, H.-J. Zeng, H. Li, Q. Yang, and Z. Chen. Enhancing text clustering by leveraging Wikipedia semantics. In *Proceedings of SIGIR'2008*, 2008.

[49] The Internet Archive. `http://archive.org/`.

[50] A. Jatowt, K. Kanazawa, S. Oyama, and K. Tanaka. Supporting analysis of future-related information in news archives and the web. In *Proceedings of JCDL'2009*, 2009.

[51] A. Jatowt, Y. Kawai, and K. Tanaka. Temporal ranking of search engine results. In *Proceedings of WISE'2005*, 2005.

[52] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of KDD'2002*, 2002.

[53] R. Jones and F. Diaz. Temporal profiles of queries. *ACM Trans. Inf. Syst.*, 25, July 2007.

[54] P. J. Kalczynski and A. Chou. Temporal document retrieval model for business news archives. *Inf. Process. Manage.*, 41, 2005.

[55] A. C. Kaluarachchi, A. S. Varde, S. Bedathur, G. Weikum, J. Peng, and A. Feldman. Incorporating terminology evolution for query translation in text retrieval with association rules. In *Proceedings of CIKM'2010*, 2010.

[56] A. C. Kaluarachchi, A. S. Varde, J. Peng, and A. Feldman. Intelligent time-aware query translation for text sources. In *Proceedings of AAAI'2010*, 2010.

[57] N. Kanhabua, R. Blanco, and M. Matthews. Ranking related news predictions. In *Proceeding of SIGIR'2011*, 2011.

[58] N. Kanhabua and K. Nørvåg. Improving temporal language models for determining time of non-timestamped documents. In *Proceedings of ECDL'2008*, 2008.

[59] N. Kanhabua and K. Nørvåg. Using temporal language models for document dating. In *Proceedings of ECML PKDD'2009*, 2009.

[60] N. Kanhabua and K. Nørvåg. Determining time of queries for re-ranking search results. In *Proceedings of ECDL'2010*, 2010.

[61] N. Kanhabua and K. Nørvåg. Exploiting time-based synonyms in searching document archives. In *Proceedings of JCDL'2010*, 2010.

[62] N. Kanhabua and K. Nørvåg. Quest: Query expansion using synonyms over time. In *Proceedings of ECML PKDD'2010*, 2010.

[63] N. Kanhabua and K. Nørvåg. A comparison of time-aware ranking methods. In *Proceeding of SIGIR'2011*, 2011.

[64] N. Kanhabua and K. Nørvåg. Time-based query performance predictors. In *Proceeding of SIGIR'2011*, 2011.

[65] M. Keikha, S. Gerani, and F. Crestani. Temper: A temporal relevance feedback method. In *Proceedings of ECIR'2011*, 2011.

[66] M. Keikha, S. Gerani, and F. Crestani. Time-based relevance models. In *Proceedings of SIGIR'2011*, 2011.

[67] J. Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of SIGKDD'2002*, 2002.

[68] A. Klose, A. Nfirnberger, R. Kruse, G. Hartmann, and M. Richards. Interactive text retrieval based on document similarities, 2000.

[69] M. Koolen, F. Adriaans, J. Kamps, and M. de Rijke. A cross-language approach to historic document retrieval. In *Proceedings of the 28th ECIR*, 2006.

[70] W. Kraaij. Variations on language modeling for information retrieval. *SIGIR Forum*, 39(1):61, 2005.

[71] A. Kulkarni, J. Teevan, K. M. Svore, and S. T. Dumais. Understanding temporal query dynamics. In *Proceedings of WSDM '2011*, 2011.

[72] N. Lathia, S. Hailes, L. Capra, and X. Amatriain. Temporal diversity in recommender systems. In *Proceeding of SIGIR'2010*, 2010.

[73] V. Lavrenko and W. B. Croft. Relevance based language models. In *Proceedings of SIGIR'2001*, 2001.

[74] X. Li and W. B. Croft. Time-based language models. In *Proceedings of CIKM'2003*, 2003.

[75] X. Li and W. B. Croft. Improving novelty detection for general topics using sentence level information patterns. In *Proceedings of CIKM'2006*, 2006.

[76] Y. Li, W. P. R. Luk, K. S. E. Ho, and F. L. K. Chung. Improving weak ad-hoc queries using wikipedia asexternal corpus. In *Proceedings of SIGIR'2007*, 2007.

[77] T.-Y. Liu. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*, 3(3):225–331, 2009.

[78] D. M. Llidó, R. B. Llavori, and M. J. A. Cabo. Extracting temporal references to assign document event-time periods. In *Proceedings of DEXA'2001*, 2001.

[79] K. E. Lochbaum and L. A. Streeter. Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Inf. Process. Manage.*, 25(6):665–676, 1989.

[80] The LongRec project. `http://research.idi.ntnu.no/longrec/`.

[81] C. Macdonald and I. Ounis. Searching for expertise: Experiments with the voting model. *Comput. J.*, 52(7):729–748, 2009.

[82] I. Mani and G. Wilson. Robust temporal processing of news. In *Proceedings of ACL'2000*, 2000.

[83] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 999.

[84] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[85] M. Matthews, P. Tolchinsky, R. Blanco, J. Atserias, P. Mika, and H. Zaragoza. Searching through time in the new york times. In *HCIR Workshop on Bridging Human-Computer Interaction and Information Retrieval, HCIR'2010*, 2010.

[86] O. Medelyan, D. N. Milne, C. Legg, and I. H. Witten. Mining meaning from Wikipedia. *Int. J. Hum.-Comput. Stud.*, 67(9):716–754, 2009.

[87] D. Metzler, R. Jones, F. Peng, and R. Zhang. Improving search relevance for implicitly temporal queries. In *Proceedings of SIGIR'2009*, 2009.

[88] D. N. Milne, I. H. Witten, and D. M. Nichols. A knowledge-based search engine powered by wikipedia. In *Proceedings of CIKM'2007*, 2007.

[89] J. Mothe and L. Tanguy. Linguistic features to predict query difficulty - a case study on previous trec campaigns. In *Proceedings of SIGIR Workshop on Predicting Query Difficulty - Methods and Applications*, SIGIR'2005, 2005.

[90] V. Murdock. *Exploring Sentence Retrieval*. VDM Verlag Dr. Mueller e.K., 2008.

[91] Mediawiki dump tool. `http://www.mediawiki.org/wiki/Mwdumper/`.

[92] NewsLibrary search engine. `http://www.newslibrary.com/`.

[93] K. Nørvåg. Supporting temporal text-containment queries in temporal document databases. *Journal of Data & Knowledge Engineering*, 49(1):105–125, 2004.

[94] S. Nunes, C. Ribeiro, and G. David. Using neighbors to date web documents. In *Proceedings of WIDM '2007*, 2007.

[95] S. Nunes, C. Ribeiro, and G. David. Use of temporal expressions in web search. In *Proceedings of ECIR'2008*, 2008.

[96] New York Times Annotated Corpus.
`http://www.corpus.nytimes.com`.

[97] Open source natural language software.
`http://opennlp.sourceforge.net/`.

[98] M. J. Pazzani and D. Billsus. Content-based recommendation systems. In P. Brusilovsky, A. Kobsa, and W. Nejdl, editors, *The adaptive web*, pages 325–341. Springer-Verlag, Berlin, Heidelberg, 2007.

[99] J. Peng, C. Macdonald, and I. Ounis. Learning to select a ranking function. In *Proceedings of ECIR'2010*, 2010.

[100] J. Perkiö, W. Buntine, and H. Tirri. A temporally adaptive content-based relevance ranking algorithm. In *Proceedings of SIGIR'2005*, 2005.

[101] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proceedings of SIGIR'1998*, 1998.

[102] Recorded Future. `https://www.recordedfuture.com/`.

[103] S. E. Robertson and K. S. Jones. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3):129–146, 1976.

[104] S. E. Robertson and S. Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of SIGIR'1994*, 1994.

[105] M. Sanderson. Ambiguous queries: test collections need more sense. In *Proceedings of SIGIR'2008*, 2008.

[106] N. Sato, M. Uehara, and Y. Sakai. Temporal ranking for fresh information retrieval. In *Proceedings of IRAL'2003*, 2003.

[107] R. Schenkel, F. M. Suchanek, and G. Kasneci. YAWN: A semantically annotated Wikipedia XML corpus. In *Proceedings of BTW'2007*, 2007.

[108] F. Scholer, H. E. Williams, and A. Turpin. Query association surrogates for web search: Research articles. *J. Am. Soc. Inf. Sci. Technol.*, 55:637–650, May 2004.

[109] R. P. Schumaker and H. Chen. Textual analysis of stock market prediction using breaking financial news: The AZFin text system. *ACM Trans. Inf. Syst.*, 27:12:1–12:19, March 2009.

[110] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated subgradient solver for SVM. In *Proceedings of ICML'2007*, 2007.

[111] B. Shaparenko, R. Caruana, J. Gehrke, and T. Joachims. Identifying temporal patterns and key players in document collections. In *Proceedings of TDM*, 2005.

[112] Y. Song, S. Pan, S. Liu, M. X. Zhou, and W. Qian. Topic and keyword re-ranking for lda-based topic modeling. In *Proceeding of CIKM'2009*, 2009.

[113] J. Strötgen, M. Gertz, and C. Junghans. An event-centric model for multilingual document similarity. In *Proceeding of SIGIR'2011*, 2011.

[114] SuperSense tagger. `http://sourceforge.net/projects/supersensetag/`.

[115] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to rank answers on large online QA collections. In *Proceedings of ACL-HLT'2008*, 2008.

[116] R. Swan and J. Allan. Extracting significant time varying features from text. In *Proceedings of CIKM'1999*, 1999.

[117] R. Swan and D. Jensen. Timemines: Constructing timelines with statistical models of word usage. In *Proceedings of KDD Workshop on Text Mining, KDD'2000*, 2000.

[118] N. Tahmasebi, K. Niklas, T. Theuerkauf, and T. Risse. Using word sense discrimination on historic document collections. In *Proceedings of JCDL'2010*, 2010.

[119] TARSQI Toolkit. `http://www.timeml.org/site/tarsqi/toolkit/`.

[120] Terrier IR Platform. `http://terrier.org/`.

[121] Time Expression Recognition and Normalization. `http://timex2.mitre.org/tern.html`.

[122] Stanford Topic Modeling Toolbox. `http://nlp.stanford.edu/software/tmt/`.

[123] V. Vinay, I. J. Cox, N. Milic-Frayling, and K. Wood. On ranking the effectiveness of searches. In *Proceedings of SIGIR'2006*, 2006.

[124] P. Wang, J. Hu, H.-J. Zeng, L. Chen, and Z. Chen. Improving text classification by using encyclopedia knowledge. In *Proceedings of ICDM'2007*, 2007.

[125] X. Wang and A. McCallum. Topics over time: a non-markov continuous-time model of topical trends. In *Proceedings of KDD'2006*, 2006.

[126] Y. Wang, M. Zhu, L. Qu, M. Spaniol, and G. Weikum. Timely YAGO: harvesting, querying, and visualizing temporal knowledge from wikipedia. In *Proceedings of EDBT'2010*, 2010.

[127] The Wayback Machine. `http://wayback.archive.org/web/`.

[128] X. Wei and W. B. Croft. LDA-based document models for ad-hoc retrieval. In *Proceedings of SIGIR'2006*, 2006.

[129] English Wikipedia completed dump (2008/01/03). http://www.archive.org/details/enwiki-20080103/.

[130] English Wikipedia snapshots. http://sourceforge.net/projects/wikipedia-miner/files/.

[131] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition*. Morgan Kaufmann, 2005.

[132] D. Wu, G. P. C. Fung, J. X. Yu, and Q. Pan. Stock prediction: an event-driven approach based on bursty keywords. *Frontiers of Computer Science in China*, 3(2):145–157, 2009.

[133] F. Wu and D. S. Weld. Autonomously semantifying Wikipedia. In *Proceedings of CIKM'2007*, 2007.

[134] Y. Xu, G. J. Jones, and B. Wang. Query dependent pseudo-relevance feedback based on wikipedia. In *Proceedings of SIGIR'2009*, 2009.

[135] P. S. Yu, X. Li, and B. Liu. On the temporal dimension of search. In *Proceedings of the 13th WWW on Alternate track papers & posters*. ACM, 2004.

[136] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proceedings of SIGIR'2007*, 2007.

[137] H. Zaragoza, H. Rode, P. Mika, J. Atserias, M. Ciaramita, and G. Attardi. Ranking very many typed entities on wikipedia. In *Proceedings of CIKM'2007*, 2007.

[138] T. Zesch, I. Gurevych, and M. Mühlhäuser. Analyzing and accessing Wikipedia as a lexical semantic resource. In *Proceedings of Biannual Conference of the Society for Computational Linguistics and Language Technology*, 2007.

[139] C. Zhai and J. Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Trans. Inf. Syst.*, 22(2):179–214, 2004.

[140] R. Zhang, Y. Chang, Z. Zheng, D. Metzler, and J.-y. Nie. Search result re-ranking by feedback control adjustment for time-sensitive query. In *Proceedings of NAACL'2009*, 2009.

[141] R. Zhang, Y. Konda, A. Dong, P. Kolari, Y. Chang, and Z. Zheng. Learning recurrent event queries for web search. In *Proceedings of EMNLP'2010*, 2010.

[142] T. Zhang. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of ICML'2004*, 2004.

[143] Y. Zhao, F. Scholer, and Y. Tsegay. Effective pre-retrieval query performance prediction using similarity and variability evidence. In *Proceedings of ECIR'2008*, 2008.

[144] Y. Zhou and W. B. Croft. Query performance prediction in web search environments. In *Proceedings of SIGIR'2007*, 2007.