REWARDING THE LOCATION OF TERMS IN SENTENCES TO ENHANCE PROBABILISTIC INFORMATION RETRIEVAL

BAIYAN LIU

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF ARTS

GRADUATE PROGRAM IN INFORMATION SYSTEMS AND TECHNOLOGY

YORK UNIVERSITY

TORONTO, ONTARIO

SEPTEMBER 2016

Abstract

In most traditional retrieval models, the weight (or probability) of a query term is estimated based on its own distribution or statistics. Intuitively, however, the nouns are more important in information retrieval and are more often found near the beginning and the end of sentences. In this thesis, we investigate the effect of rewarding the terms based on their location in sentences on information retrieval. Particularly, we propose a kernel-based method to capture the term placement pattern, in which a novel Term Location retrieval model is derived in combination with the BM25 model to enhance probabilistic information retrieval. Experiments on five TREC datasets of varied size and content indicates that the proposed model significantly outperforms the optimized BM25 and DirichletLM in MAP over all datasets with all kernel functions, and excels compared to the optimized BM25 and DirichletLM over most of the datasets in P@5 and P@20 with different kernel functions.

Acknowledgements

I would like to thank my supervisor Dr. Jimmy Huang for his motivation and guidance. I would like to thank my thesis committee member Dr. Augustine Wong for his insight and suggestions and my defense committee member Dr. Aijun An for her thorough scrutiny. I would also like to thank Dr. Jeff Ye and Dr. Xiangdong An from the Information Retrieval and Knowledge Management lab for their help in designing and running my experiments.

Finally, I would also like to thank my wife Anjie for her support.

Contents

Chapter 1

Introduction

1.1 History of Information Retrieval

Information retrieval (IR) is simply the concept of cataloging and retrieving records within a collection, such as web pages on the Internet, books in a library, or chapters in a book. IR originated from librarianship. It arose from the need to manage large amounts of documents efficiently in such a way that facilitates retrieval. Historically, written records were ordered categorically, alphabetically, or catalogued with a table of contents. A well-known example of a table of contents is the first book of *Natural History*, a comprehensive encyclopedia published by Pliny the Elder circa 77-79 AD. In addition, Pliny the Elder credits Quintus Valerius Soranus, a Latin poet, for his precedence in creating a table of contents approximately a hundred years before in *On Mysteries*. However, more primitive methods of cataloging can be traced back to around 3000 BC [1].

Historic methods of manually cataloging and retrieving records are not sufficient for modern information. The advent of and the advances in computers followed by the Internet have led to what's known as information overload, where data is produced quicker than our ability to ingest it. Vannevar Bush first planted the idea of using computers to perform automatic IR in *As We May Think* in 1945 [2]. The next three decades then saw the building of the foundation of modern IR.

Formally, modern IR, or text retrieval, is the task of identifying the documents in a document collection that are relevant to the information needs. The relevant documents

are often ranked by the degree of relevance, from the most relevant document to the least relevant document. Information needs is expressed in the form of a query, such as a search string in a web search engine. The documents are often indexed or catalogued based on the terms in the document, which is based on Luhn's work [3]. The degree of relevance can then be determined by calculating the degree of similarity between the query and the document. Many IR models have been developed to calculate this degree of similarity.

1.2 Information Retrieval Models

1.2.1 The Boolean Model

The earliest IR models are the Boolean models, which are based on set theory. In the standard Boolean model, the document D is represented as a set:

$$D = \{d_1, d_2 \dots d_n\} \tag{1.1}$$

$$\text{where } d_i = \begin{cases} 1 & \text{if } t_i \in D \\ 0 & \text{if } t_i \notin D \end{cases} \tag{1.2}$$

$$\text{and } T = \{t_1, t_2 \dots t_n\} \tag{1.3}$$

where T is the set of all terms used in the IR system, t_i is the ith term in T, and n is the number of terms in T. The query Q can only be formulated as a Boolean expression with the Boolean operators AND, OR, and NOT. The document is considered relevant only if it is an exact match to Q, otherwise, the document is not considered relevant. The limitations of the standard Boolean model is that it is not capable of ranking documents, partially matching documents, or assigning different weights to terms. Complex Boolean expressions are also difficult to formulate.

1.2.2 The Vector Space Model

The vector space model [4–6] is based on linear algebra. It improves upon the rigidness of the standard Boolean model. In the vector space model, the document D and the

query Q are represented as vectors:

$$\overrightarrow{D} = \{d_1, d_2 \dots d_n\} \text{ and } \overrightarrow{Q} = \{q_1, q_2 \dots q_n\} \tag{1.4}$$

where d_i and q_i are the weights of the term t_i in D and Q, respectively, t_i is the ith term in T, and n is the number of items in T. T is defined in Equation 1.3. The weight of a term determines the importance of the term in the document.

A typical way of calculating the weight of t_i in D in the vector space model is with term statistics based approaches such as the $TF\text{-}IDF$ weighting model [5]. The $TF\text{-}IDF$ weighting model is based on two components, TF, which promotes terms that appear frequently in the document, and IDF, which demotes terms that appear frequently in the document collection. The degree of similarity between D and Q, $sim(D, Q)$, is calculated with the cosine similarity measure as follows:

$$sim(D, Q) = \frac{\sum_{i=1}^{n}(d_i * q_i)}{\sqrt{\sum_{i=1}^{n} d_i^2} * \sqrt{\sum_{i=1}^{n} q_i^2}} \tag{1.5}$$

The main limitation of the vector space model is that the semantic and statistical dependence information between the terms in the document is lost. This is because the vector representation assumes that the dimensions in the vector are independent. In IR, assuming that the terms in the document are semantically and statistically independent is also known as the bag of words approach.

1.2.3 Probabilistic Models

The probabilistic relevance model [7] is based on probability theory. It was proposed in 1976, around the same time as the vector space model. In the probabilistic relevance model, the document D and the query Q are represented as vectors. D and Q are defined in Equation 1.4. The degree of similarity between D and Q, $sim(D, Q)$, is estimated as follows:

$$sim(D, Q) = \frac{P(R|D)}{P(\overline{R}|D)} = \frac{P(D|R)P(R)}{P(D|\overline{R})P(\overline{R})} \tag{1.6}$$

where R is relevance and $\overline{R}$ is non-relevance. $P(R|D)$ is then the probability that D is relevant and $P(\overline{R}|D)$ is the probability that D is not relevant. Since for a single query, $P(R)$ and $P(\overline{R})$ are constant for every document in the document collection, equation

1.6 can be formulated as follows:

$$sim(D,Q) \stackrel{rank}{=} \frac{P(D|R)}{P(D|\overline{R})} \tag{1.7}$$

Several weighting models have been proposed to estimate Equation 1.7. The most well-known and effective models are the language model [8–10] and the BM25 model [11–14]. Both the language model and the BM25 model are state of the art weighting models in IR. However, the probabilistic relevance model and the vector space model are not dissimilar. The main difference is the way that the degree of similarity between a document and a query is calculated. The probabilistic relevance model also assumes that all the terms in the document are semantically and statistically independent.

1.2.3.1 The Language Model

The language model was proposed by Ponte and Croft in 1998 [8]. The model generates a language model, or a probability distribution, M_D from the document D. It then estimates the probability that the query Q is generated by D. The documents are ranked by the probability that Q is generated from each document, from the mostly probable to the least probable. The uni-gram language model follows the bag of words approach and is the most common. In the uni-gram language model, the probability that Q is generated by D, $P(Q|M_D)$, is as follows:

$$\text{given } Q = \{q_1, q_2 \dots q_n\} \tag{1.8}$$

$$P(Q|M_D) = P(q_1, q_2 \dots q_n|M_D) = \prod_{i=1}^{n} P(q_i|M_D) \tag{1.9}$$

where q_i is the ith term of Q. Smoothing is often used to ensure that $P(q_i|M_D) > 0$. In our experiments, we compare the performance of our model against the DirichletLM model, which is the uni-gram language model with Dirichlet smoothing [15].

1.2.3.2 The BM25 Model

The BM25 model was proposed by Robertson et al in 1995 [12]. It is based on the 2-Poisson model [16]. The 2-Poisson model assumes that there are two types of terms,

function terms and specialty terms. The distribution of a function term in a document collection can be modeled with a Poisson distribution. However, they discovered that there exists two levels of treatment for specialty terms in a document collection. This indicates that there are two document classes for each specialty term. Modelling the distribution of a specialty term in a document collection then requires a 2-Poisson distribution, which is defined as follows:

$$f(t=k) = \pi \frac{e^{-\lambda_1}\lambda_1^k}{k!} + (1-\pi)\frac{e^{-\lambda_2}\lambda_2^k}{k!} \tag{1.10}$$

where λ_1, λ_2, and π are parameters. λ_i is the mean frequency of the term t in the document class i, π is the proportion of class 1 documents in the document collection, and $f(t=k)$ is distribution of t with frequency k in the document collection.

In 1980, Robertson et al [17] proposed a probabilistic weighting model. In this model, the weight score of the document D, $w(D)$, is defined as follows:

$$w(D) = log\frac{P(D|R)P(\underline{0}|\overline{R})}{P(D|\overline{R})P(\underline{0}|R)} \tag{1.11}$$

where $\underline{0}$ is the zero vector. In 1994, Robertson and Walker combined Equation 1.10 with Equation 1.11 and proposed the BM11 and BM15 models [11]. In 1995, Robertson et al then combined BM11 and BM15 into the BM25 model [12]. The contribution of BM11 and BM15 into BM25 is controlled with the b parameter. BM25 is equivalent to BM15 if $b = 0$ and is equivalent to BM11 if $b = 1$. In our experiments, we extend the BM25 model and compare the performance of our model against the BM25 model.

1.3 Natural Language Processing

Most modern IR models such as the DirichletLM and the BM25 models use the bag of words approach. However, the bag of words approach is only used to simplify the models and the assumption that all the terms in a document are independent is not always true in practice. For example, the terms “rock” and “star” in a document could indicate that the document is about astronomy, but if the terms are adjacent to one another, then the document could be about music. However, the bag of words approach treats

both scenarios equivalently. Therefore, with the bag of words approach, all semantic and statistical relations between the terms in a document are lost.

Natural Language Processing (NLP) is the concept of extracting the semantic meaning from natural language. NLP has a wide range of applications, such as machine translation, automatic summarizing, and IR. For example, stemming is a NLP task. Stemming is the process of reducing a term to its stem such that all morphological variants of the term are represented identically. For example, both "connecting" and "connected" would be stemmed to "connect". In most modern IR systems the terms in the document and the query are stemmed. In our experiments, we use the English Porter stemmer [18], which is based on suffix stripping.

1.3.1 Statistical Phrases

NLP can be used improve IR by replenishing the semantic meaning removed by the bag of words approach. A common and effective NLP approach to improve IR is to combine the terms into phrases. There are two types of phrases, statistical phrases and syntactical phrases [19]. Statistical phrases are non-function terms that often appear together in a document. For example, if the term "rock" is always immediately followed by the term "star" in a document, then "rock star" is a statistical phrase in the document. Statistical phrases are not limited to two terms. However, bi-gram statistical phrases are the easiest to implement and the least complex to compute. The effectiveness of higher order n-grams also suffer from diminishing returns since the frequency of higher order n-grams is much lower than bi-grams. Low frequency n-grams are not good document discriminators and have an adverse effect on retrieval recall [5]. Therefore bi-gram statistical phrases provide the best trade-off between effectiveness and complexity and are the most common.

However, strict adjacency is too rigid of a requirement since two semantically related terms may not always be adjacent. For example, in Figure 1.1, "ice" is semantically related to "asteroid" but is not always adjacent to "asteroid".

Scientists have discovered water ice on an asteroid for the second time. The asteroid's ice layer is probably less than one micron thick.

FIGURE 1.1: Example: semantically related non-adjacent terms

Both query accuracy and query coverage is needed for good retrieval performance [5, 19]. Enforcing strict adjacency may be too restrictive and may have an adverse effect on query coverage and therefore retrieval recall. A more effective approach is to estimate the probability that two terms are semantically related by analyzing the distance between the occurrences of the two terms in a document. Two terms that always appear close together in a document are more likely to be semantically related. Many IR models have been proposed to estimate this probability using methods such as a sliding window and kernel functions [20–25].

1.3.2 Syntactic Phrases

Syntactic phrases are sets of terms that follow syntactic patterns. For example, 'blue hat" matches the syntactic pattern {adjective noun} and is a syntactic phrase. Part-of-speech tagging is performed to identify the part-of-speech that each term in the document corresponds to. Interesting syntactic patterns are then extracted from the document and used in IR. Noun-based phrases, or noun-phrases, have been found to be the most effective type of syntactic phrases in improving retrieval performance [26]. However, extracting syntactic phrases from the documents is more computationally intensive than extracting statistical phrases, but can be more effective [19].

We want to combine the effectiveness of syntactic phrases with the simplicity of statistical phrases. Specifically, we want to identify the nouns in the documents using statistical phrase extraction methods in order to improve probabilistic retrieval performance.

1.4 Sentence Patterns

In linguistic topology, languages can be classified by the most common word-order of sentences. English is classified as a subject-verb-object (SVO) language [27]. This means that in most English sentences, the subject is placed at the beginning of the sentence and the object is placed at the end of the sentence.

Most English sentences follow the six basic sentence patterns shown in Table 1.1 [28]. In most of the sentence patterns, the subject is placed at the beginning of the sentence. The subject, direct object, subject complement, or object complement is placed at the

	Subject	Predicate		
Pattern 1	Subject	Intransitive verb		
	The baby	is sleeping.		
Pattern 2	Subject	Linking verb	Subject complement	
	Dogs	are	social animals.	
Pattern 3	Subject	Transitive verb	Direct object	
	We	visited	our aunt.	
Pattern 4	Subject	Transitive verb	Direct object	Object complement
	Our neighbors	leave	their dogs	alone.
Pattern 5	Subject	Transitive verb	Indirect object	Direct object
	They	bought	her	a new leash.
Pattern 6	There or It	Verb (usually be)	Subject	
	There	isn't	any hot water.	

TABLE 1.1: Basic English sentence patterns [28]

end of the sentence. The subject and direct object are comprised of nouns and noun-phrases. The subject complement and object complement are comprised of adjectives or nouns. These sentence patterns suit most but not all English sentences, and exceptions are rare [27].

Google bid, but lost the auction to own the Nortel patents.

FIGURE 1.2: Example: noun placement pattern in sentences

Modern methods of integrating part-of-speech information into IR involve part-of-speech tagging or with the assistance of a dictionary or a thesaurus. These methods are complex and computationally intensive. We want to integrate basic English sentence patterns information into IR in order to identify the nouns in the documents. We do this with statistical phrase extraction methods in order to avoid the cost in complexity and computing time associated with traditional methods. Specifically, we want to estimate the probability that a term is a noun based on the location of the term in sentences. Since the subject and the object are placed at the beginning and the end of most of sentences, a term that always appear at the beginning or the end of sentences is more likely to be a noun.

1.5 Term Proximity

Term proximity in IR is the technique of measuring the distance between the terms in a document to estimate the semantic relatedness of the terms. This is to extract statistical phrases from documents. Many IR models have been proposed to meaningfully capture this information. Most of the well-known models are based on adjacency [19], usually within a window, spans [23], and the term-to-term distance [24]. However, span-based measures such as *Span* [29] and $MinCover$ [23] are not as effective as the other measures. Indeed, the density of spans was found to be negatively correlated with the relevance of the documents [25].

We use the location of the term in sentences to estimate the probability that the term is a noun. There are two methods that we can use to capture the location of the terms. The first method is to measure the distance between the terms and the nearest sentence-final punctuation. In doing this, we can apply term proximity measures to measure the proximity of the term to the beginning or the end of sentences. We can then use this information to estimate the probability that the term is a noun. The second method is to measure the distance between the terms and the middle of sentences. We can then estimate the probability that the term is not a noun. In our experiments, we choose the latter method using a combination of the adjacency within a window and the term-to-term distance measures.

In order to integrate this information into IR, we reward a term t_D based on the probability that t_D is a noun in the document D, where t is a term in the query. t_D is given more reward if t_D is more likely to be a noun. We do this for all of the documents in the document collection in order to promote the documents where t is more likely to be used as a noun and demote the documents where t is less likely to be used as a noun. Nouns are more important than the other part-of-speech in a document in IR [26]. Therefore we hope to elevate the documents in which t is important and improve probabilistic IR.

The main contributions of this thesis are summarized as follows:

- We investigate the importance of the patterns of terms in the English language.

- In order to reward terms that are more likely to be nouns, we propose a kernel-based method to capture the term placement pattern, in which we propose a novel Term Location retrieval model.
- A normalization is proposed specifically for balancing term weights in short and long sentences.

The rest of this thesis is organized as follows. In Chapter 2, we discuss related work in integrating syntactic patterns and term proximity information into IR. In Chapter 3, we conduct preliminary experiments to prove the soundness of our assumptions. In Chapter 4, we show the details of the implementation of our model and the effect of the parameters in our model. In Chapter 5, we show the settings under which we conduct our experiments and perform our evaluation. In Chapter 6, we show and analyze the results of our experiments. In Chapter 7 we present our conclusions and discuss possible extensions to our model.

Chapter 2

Related Work

2.1 Syntactic Nouns and Noun-Phrases

In IR, the weighting model is one of the most important components. The weighting model is used to assign scores to the documents according to their degree of similarity to a given search query. In the past decades, a large number of models such as the TF-IDF weighting model [5], BM25 [12], and the language model [8] have been proposed to address this problem. Most of the models estimate the importance (or probability) of a query term in a document based on the distribution or the statistics of the term, such as the term frequency, the collection term frequency, and the document frequency. However, as described in Chapter 1.2, most of the models use the bag of words approach to accomplish this. Statistically similar terms and documents may not be semantically similar. For example, the terms "blue hat" and "red hat" are statistically similarly, but could be unrelated semantically. Similarly, "red hat" could also be semantically unrelated to the term "hat".

In order to refine this estimation, some researchers have proposed to identify nouns and noun-phrases in the documents using NLP. Jing and Croft [26] found that previous attempts at automatically constructing association thesauri were not very successful in improving IR. They theorized that one of the causes was that all lexical categories were treated equally. They proposed PhraseFinder to automatically construct collection-dependent association thesauri. They then used the association thesauri to assist query expansion and found that nouns and noun-phrases were the most effective in improving

IR. Evans and Zhai [30] recognized that using single words as indexing terms, or the bag-of-words approach, may not accurately describe the document and is prone to many problems, such as the phrase normalization problem. The phrase normalization problem is where phrases that are syntactically different but semantically similar, such as "junior college" and "college junior", are matched. In order to address those problems, they extended the phrase-based indexing system in CLARIT to include the following noun-based phrase types: lexical atoms, head modifier strips, sub-compounds, and cross-preposition modification pairs. Liu et al [31] classified noun-phrases into four types, proper names, dictionary phrases, simple phrases, and complex phrases and ranked documents based on phrase similarity. They also imposed additional constraints on candidate term selection in pseudo relevance feedback. Terms also need to be either highly positively globally correlated with a query term or phrase or contain query terms in its WordNet definition to be chosen as a candidate. Zheng et al [32] used noun-phrases and semantic relationships to represent documents in order to assist document clustering. Noun-phrases were extracted with the assistance of WordNet. There are also other work that used nouns and noun-phrases to improve IR [19, 33, 34].

However, previous work that use nouns and noun-phrases to improve IR rely on a thesaurus such as WordNet or a part-of-speech tagger to facilitate the identification of the nouns and noun-phrases. These methods, while effective at replenishing the semantic information in the documents, are computationally intensive. We propose to use the positional information of the terms in sentences to calculate the probability that the terms are important. Our proposed method does not rely on a thesaurus or a part-of-speech tagger, which should alleviate the cost in computing time associated with those approaches.

2.2 Syntactic Patterns

A syntactic pattern is a linguistic pattern or schema of the semantic relationships between terms. For example, {adjective noun} is a syntactic pattern. Sentence patterns is also a type of syntactic pattern. Many part-of-speech taggers such as the Brill tagger [35] rely on the usage of syntactic patterns to correctly tag the terms in sentences. Similarly, syntactic patterns have been used to facilitate the identification of the nouns and noun-phrases, which are the important terms in the documents in IR.

Yang et al [36] used a parse tree to transform sentences in legal agreements into SVO representations of sentences. The SVO representations are then used in conjunction with cue terms to identify the provisions provided by sentences, such as assignment and risk of loss. In their experiments they found that provisions extraction using the SVO representation resulted in high precision but low recall, which could be due to the specificity of SVO sentence patterns and the difficulty in parsing complex sentences. Liu et al [37] used syntactic patterns such as {NP such as NP} and {NP aka NP} to extract semantic relationships such as meronymy and synonymy from clinical documents. However, experimental results varied between the two corpora used and not all syntactic patterns produced the same quality of semantic relationships. They also observed a low recall rate in their experiments due to the low frequency of the syntactic patterns in some of the documents. Hung et al [38] used syntactic pattern matching to extract syntactically complete sentences that express event-based commonsense knowledge from web documents. Semantic role labeling is then used to tag the semantic roles of the terms in sentences, such as the subject and the object. Various plausibility verification heuristics are then applied to sentences to prune the low quality sentences. The effectiveness of their approach was comparable to ConceptNet, a human annotated commonsense knowledge base. Ibekwe-SanJuan et al [39] built finite state automaton with syntactic patterns and synonyms from WordNet. The finite state automaton is used to tag sentences in scientific documents according to its category, such as result and objective. The tagged sentences are then used to summarize the documents.

The above work are all limited in scope. The experiments were only conducted on a specific type of documents or sentences, for example, legal documents [36] and scientific documents [39]. This is due to the difficulty of training or finding all effective syntactic patterns in a corpora [39]. Likewise, it is also difficult to find syntactic patterns that are effective in all the documents of a single corpus [37]. Rule-based part-of-speech taggers are less effective in unseen text. Therefore, we propose to use sentence patterns. Since English is a SVO language [27], most of the English sentences follow the syntactic pattern {subject verb object}. Yang et al [36] used the SVO syntactic pattern to extract the subject, the verb, and the object from sentences. Similarly, we propose to use the SVO syntactic pattern to calculate the probability that a term is part of the subject or part of the object. We do this based on its nearness to the start or the end of the sentence. Furthermore, [36, 37] both observed low recall in their experiments due to

the specificity of the syntactic patterns. Yang et al [36] used legal cue words to parse sentences and Liu et al [37] used syntactic patterns that only matched specific types of semantic relationships. Therefore we propose to use general sentence patterns that are applicable to most of sentences in English, which should improve recall.

2.3 Other Patterns in Documents

In addition to syntactic patterns, there exists other patterns in English documents [40–49]. Many researchers have used the pattern information in the documents to reward the terms that are more important in order to improve retrieval performance. Zhao et al [50] theorized that terms near the beginning of a document are more important in information retrieval. This is based on the intuition that a summary is a good representation of a document and that authors often summarize of their ideas near the beginning of the document. They proposed shape functions to reward terms based on its nearness to the beginning of the document. This is to use the syntactic structure information of the document and estimate the probability that a term is important. Our proposed approach is similar. However, we consider the syntactic structure information in sentences of the document instead of the whole document. Similarly, Seo and Jeon [51] re-ranked documents based on the normalized query likelihood scores of sentences in the documents. The query likelihood score of a sentence S is $P(Q|S)$, or the probability of the query Q given the uni-gram language model of S. A logical regression model is then used with various features such as the variance of the relevance levels and the position of the first peak to greatly improve the precision of the top 5 documents. Trotman [40] gave certain parts of academic documents more weight, such as the headline, the author, and the leading paragraph. The weights were trained using a genetic algorithm. This approach makes similar assumptions as [50]. However, no significant improvement over BM25 was observed. Though the context unit of [40, 50] are different from our proposed approach, the motivation is the same, which is to use the non-obvious patterns in the documents to improve IR.

There also exist works that use the structural information of the explicit structures, such as XML elements [52] and lists [41], in the documents to improve IR. Broschart and Schenkel [52] gave more weight to terms co-occurring in the same element than terms occurring close together but in different elements in XML documents. They did this by

introducing virtual gaps between the elements so that terms in different elements are considered more distant. This approach could be integrated into our proposed method but in the context of a sentence instead of a XML document. Varying-sized virtual gaps could be introduced for cue words or punctuation when calculating the distance between a term and sentence-final punctuation. Dou et al [41] used free text patterns, HTML tag patterns, and repeat region patterns in web documents to extract lists from the documents, which were clustered into dimensions. Analysis showed that the top results for each query contained many meaningful dimensions. Lists can be used to find hypernymy and meronymy relationships in the documents, which can be used to improve IR [37]. This extension is beyond the scope of this thesis. We leave them for future investigation. Zhou et al [45] explored the effect of document length on document relevance. They used kernel density estimation to estimate the probability density of the relevant and non-relevant documents in a collection. Data transformation algorithms is then applied on top of that in order to better visualize the difference between relevant and non-relevant documents, from which a distribution function is estimated for the relevant and non-relevant documents. This information is then integrated into BM25 in order to significantly improve it.

2.4 Term Proximity

Since we strive to achieve the same effect as syntactic phrase extraction but with simpler methods. we turn to statistical phrase extraction methods, which are less complex and less computationally intensive. A popular and well-studied approach is to use term proximity to estimate the probability that the statistical phrase comprised of two terms is related, where the probability is higher if the two terms are closer together. Many models have been proposed to measure the distance between the terms and estimate that probability [53–55].

Liu et al proposed a similar model in [56]. Rasolofo and Savoy [20] found that using term proximity information with Okapi BM25 improved retrieval performance especially with top documents. They define distance as the number of terms between two key terms. Büttcher et al [21] extended [20] and processed the queries using the document-at-a-time approach. They defined the distance between key terms T_i and T_j as the number of postings between posting P_i, containing T_i, and posting P_j, containing T_j. They

found that term proximity information is more important in larger collections. They theorized that the effect is due to term proximity information effectively discriminating against the increased number of non-relevant documents that contain the query terms in larger collections. Tao and Zhai [23] studied span-based distance measures and aggregated pairwise distance measures to measure term proximity. In their experiment, the span-based distance measures were not effective but the aggregated pairwise distance measure $MinDist$ improved retrieval performance significantly. They also found that term proximity information is more effective in longer documents. He et al [57] used sliding windows and survival analysis to model the proximity between terms and extended BM25. Song et al [58] used devised a method to group close query terms together with no overlap. Their approach was more consistently precise than [20]. Cho et al [59] used Bahadur-Lazarsfeld expansion to generate term pairs within a window. While improvements were moderate overall, large improvements were seen in precision in top documents. However, they cited performance issues since the pairs were generated at retrieval time, and suggested a database with pre-calculated pairs to account for this. Miao et al [60] integrated the term proximity information between the terms in the query and the candidate expansion terms in Rocchio's pseudo relevance feedback model. Term proximity was calculated using either a sliding window, the Gaussian kernel function, or the Hyperspace Analogue to Language (HAL) method. All three methods significantly improved BM25 with Rocchio's. However, the HAL method proved to be the most effective.

From those experiments, span-based measures do not seem to be effective in measuring term proximity. Window-based and aggregated pairwise distance measures [23] are more effective. In this thesis, we use a window-based method and $AvgDist$ to measure term proximity.

2.4.1 Kernel Functions

Kernel functions is another method of measuring term proximity. Kernel functions are flexible. A term is not either inside (1) or outside (0) of a window, instead a value between [0, 1] is given according to the distance between the two terms. Kernel functions have been proven to be highly effective in improving IR.

Beigbeder and Mercer [61] integrated kernel functions into the Boolean model to approximate term co-occurrence. Petkova and Croft [22] used kernel density estimation $\sum_i^N k(t,c)$ to estimate the strength of association between query terms and candidate entities $p(t|c,d)$. $p(t|c,d)$ is the probability that the query term t is associated with the candidate entity c in the document d. This was used to estimate $p(c,t)$ in order to perform named entity retrieval. Zhao et al [24] proposed the CRTER model, which used kernel density estimation to estimate the probability that two closely occurring terms form a bi-gram phrase. They originated kernel functions from key terms, where the probability of the terms forming a bi-gram phrase is the value of the kernel function where two kernel functions intersect. They used Gaussian, Triangle, Uniform, Circle, and Cosine kernel functions from [62] and additionally introduced the Quartic, Epanechnikov, and Triweight kernel functions to IR. Zhu et al [25] introduced *Span_Cover* to measure the distance between two query terms. *Span_Cover* is a span based distance measure, similar to $MinCover$ [23]. They then used the density of each *Span_Cover* instance in a document, which were calculated with a kernel function, and the number of *Span_Cover* instances in a document to improve probabilistic information retrieval. Barakat et al [63] integrated cross terms from [24] into the uni-gram language model.

We adapt kernel functions in term proximity to measure the distance between a term and sentence-final punctuation. Most of the English sentences follow the syntactic pattern {subject verb object} [27]. Therefore we use kernel functions to estimate the probability that a term is the subject or the object based on its distance to where we expect the subject or the object to be in a sentence. In contrast with [24], our proposed method is designed such that non-symmetrical kernel functions and kernel functions with negative values can be adopted. However, we leave that for future investigation.

2.4.2 Sentence-based Summarization

Challenges and techniques in sentence-based summarization are also applicative in sentence-based IR models. For example, Fisher and Roark [64] extracted sentences from documents and used supervised sentence rankings for extractive summarizing. They used mostly features aggregated from word-based features such as the tf-idf of a term in a cluster and sentence positions. They also removed short and long sentences to avoid simple sentences and long lists, respectively, which make poor summaries. Quotes and

anaphora resolution presented challenges for them. Luhn [65] derived the significance factor of a sentence based on the frequency of the terms in sentences of the document and the position of the terms in sentences. A feasibility experiment was then conducted by combining the most significant sentence in each section of a document into an abstract. In the paper it was hypothesized that synonyms and the authors' writing styles may pose problems to their method.

Chapter 3

Preliminary Experiments

In order to ensure that our assumptions are sound in practice, we conduct several preliminary experiments. Our main assumptions are summarized as follows:

- Nouns are more likely to be found near the beginning and the end of sentences. In Chapter 3.2 we assess the soundness of this assumption by analyzing the placement of the nouns in sentences.
- Important terms are more likely to be found near the beginning and the end of sentences. In Chapter 3.3 we assess the soundness of this assumption by analyzing the placement of the query terms in sentences of the relevant and the non-relevant documents.

Lastly, in Chapter 3.4 we conduct a final preliminary experiment to assess the effectiveness of our proposed weighting method.

3.1 Experimental Settings

We conduct our first and second preliminary experiments on the WT2g data set. WT2g is a 2 GB size crawl of general web documents. It was used in the TREC 1999 Web track. WT2g is comprised of 247,491 documents. In each document, we ignore the header by only processing the text between the </DOCHDR> and </DOC> tags. We also remove all HTML tags, numbers, and characters that the Stanford Tagger

cannot parse. In the second preliminary experiment, we use the topics 401-450 and the corresponding relevance ground truths in order to determine the relevant and the non-relevant documents. We use the Linux Mint 15 and 16 x64 and Ubuntu 12.04 x86 operating systems with Java SE 8u5 x64 and x86, respectively.

The common experimental settings for all our preliminary experiments are as follows. We use the English Stanford Tagger 3.3.1 for part-of-speech tagging and sentence identification. We add newlines to the Stanford Tagger's default set of sentence delimiters. Punctuation that are tagged as any of {# $ " ( ) , : “} are removed prior to processing. We do not analyze non-sentence-final punctuation in this thesis. We stem the terms using an English Porter stemmer [18]. Stop words are removed using the stop words list from Terrier 3.5. We also remove both the short and the long sentences in order to avoid simple sentences and long lists, respectively [64]. We only keep a sentence s if $|s| \geq 7$ and $|s| \leq 20$ where $|s|$ is the number of terms in s. The sentence length thresholds are chosen arbitrarily, we leave the in-depth analysis for a future study.

3.2 Placement of Nouns in Sentences

3.2.1 Design

This preliminary experiment is designed to analyze the placement of nouns in sentences. We define a noun as a term that is tagged as NN (noun, singular or mass), NNP (proper noun, singular), NNPS (proper noun, plural) or NNS (noun, plural).

In this experiment, we measure $AvgD$, $AvgSL$, and the number of nouns and sentences in the collection. $AvgD$ is the average of the normalized distances of the nouns from the middle of sentences. $AvgD$ is defined as follows:

$$AvgD = \frac{\sum_{t \in T} \frac{|Mid(t) - Pos(t)|}{Mid(t)}}{|T|} \tag{3.1}$$

$$\text{where } Mid(t) = \frac{SenLength(t) - 1}{2} \tag{3.2}$$

where $Pos(t)$ is the position of t in the sentence, T is the nouns in the document, $|T|$ is the number of terms in T, and $SenLength(t)$ is the length of the sentence that t is in. $AvgD$ has a range of [0, 1], where a larger $AvgD$ means that the terms are nearer

to the beginning and the end of sentences. $AvgSL$ is the average lengths of sentences in the collection and is defined as follows:

$$AvgSL = \frac{\sum_{s \in C} |s|}{m} \tag{3.3}$$

where s is a sentence in the collection C and m is the number of sentences in the collection. The algorithm we use is shown in Figure 3.1.

```
1  for (document d in collection) {
2    for (sentence s in d) {
3      if (|s| ≥ 7 && |s| ≤ 20) {
4        for (term t in s) {
5          if (Tag(t) == noun) {
6            calc_term_statistics(t);
7          }
8        }
9        calc_sen_statistics(s);
10     }
11   }
12 }
```

FIGURE 3.1: Algorithm of the noun placement preliminary experiment

Lastly, we calculate the statistics for the terms that occur in the left half of sentences and in the right half of sentences separately. This is to see if the placement of the subjects and objects in sentences are different. A term t is in the left half of the sentence if $Mid(t) - Pos(t) < 1$ and is in the right half of the sentence if $Mid(t) - Pos(t) > 1$. Occurrences of t where $Mid(t) - Pos(t) = 0$ are ignored. The source code for the noun placement parser we use in this experiment can be found in Appendix A.2.

3.2.2 Results

The results of this experiment are shown in Table 3.1 as follows:

Side	$AvgD$	# Nouns	$AvgSL$	# Sentences
Left	0.5901	24,918,926	10.5619	14,360,676
Right	0.613	26,286,542		

TABLE 3.1: Results of the noun placement preliminary experiment

$AvgD > 0.5$ for the nouns in both of the halves of sentences. This means that the nouns are nearer to the beginning and the end of sentences. Particularly, the nouns in the right half of the sentence are more distant from the middle of sentences than nouns in the left half of sentences. This shows that the subjects and the objects in sentences have distinct placement patterns. However, the difference is small.

From the results, we can conclude that our first assumption is true in at least the WT2g collection. However, we require additional experiments in order to see if the placement information of the terms in sentences can be used to distinguish between the relevant and the non-relevant documents.

3.3 Placement of Important Terms in Sentences

3.3.1 Design

This preliminary experiment is designed to analyze the placement of the query terms in the relevant and the non-relevant documents. We do this by calculating various term-based and sentence-based statistics in both the relevant and the non-relevant documents. We then look for large differences in the statistics between the relevant and the non-relevant documents. The algorithm we use is shown in Figure 3.2.

```
1  for (document d in results) {
2    for (sentence s in d) {
3      if (|s| ≥ 7 && |s| ≤ 20) {
4        rel = d.rel > 0;
5        for (term t in s) {
6          t.stem();
7          if (t in d.query_terms) {
8            calc_term_statistics(t, rel);
9          }
10       }
11       if (# query terms in s > 0) {
12         calc_sen_statistics(s, rel);
13       }
14     }
15   }
16 }
```

FIGURE 3.2: Algorithm of the query term placement preliminary experiment

We run one pass of retrieval on the WT2g collection using BM25 with $k_1 = 1.2$, $k_3 = 8$, and $b = 0.2$. We use the set of documents returned from that retrieval pass as our document collection in this experiment. We use the same $AvgD$, $AvgSL$ measures from Chapter 3.2, except T for each document is the set of corresponding query terms. Additionally, we use the $AvgP$, $AvgPN$, and $AvgPL$ measures. $AvgP$ is the average of the absolute distances of T from the beginning or the end of sentences, whichever is the nearest for each term occurrence. $AvgP$ is defined as follows:

$$AvgP = \frac{\sum_{t \in T} MinPos(t)}{|T|} \tag{3.4}$$

$$\text{where } MinPos(t) = Min(Pos(t),\, SenLength(t) - Pos(t) - 1) \tag{3.5}$$

where a smaller $AvgP$ means that the terms are nearer to the beginning or the end of sentences. $AvgPN$ and $AvgPL$ are the average of the normalized and the $\log_2$ normalized values of $AvgP$. We use the normalization scheme from [66] for $AvgPN$. $AvgPN$ and $AvgPL$ are to see if dampening the effect of the outliers will have a large effect. $AvgPN$ and $AvgPL$ are defined as follows:

$$AvgPN = \frac{\sum_{t \in T} \frac{MinPos(t)}{1+MinPos(t)}}{|T|} \tag{3.6}$$

$$AvgPL = \frac{\sum_{t \in T} log_2(MinPos(t))}{|T|} \tag{3.7}$$

We want to be able to meaningfully compare the ability of the measures to discriminate between the relevant and non-relevant documents. Therefore we divide the values from the relevant documents by the values from the non-relevant documents for each measure, for example, $AvgD = \frac{AvgD_{rel}}{AvgD_{non-rel}}$. Similar to Chapter 3.2, we also calculate the statistics for the terms that occur in the left half of sentences and in the right half of sentences separately. The source code for the query term placement parser we use in this experiment is found in Appendix B.2.

3.3.2 Results

The results of this experiment are shown in Table 3.2 as follows:

Side	$AvgD$	$AvgP$	$AvgPN$	$AvgPL$	$AvgSL$
Left	1.03	0.9523	0.9646	0.9599	0.9902
Right	1.0021	0.9881	0.9858	0.9869	

TABLE 3.2: Results of the query term placement preliminary experiment

$AvgD > 1$ and $AvgP < 1$ for the terms in both halves of sentences. This means that the terms in the relevant documents are nearer to the beginning or the end of sentences than the terms in the non-relevant documents. Particularly, the difference in the placement of the query terms in the left half of sentences is larger than in the right half of sentences. This means that the placement of the subjects in sentences are better relevance discriminators than the placement of the objects. Since $AvgPN \approx AvgP \approx AvgPL$, outliers do not seem to have a large effect. This is probably due to the fact that we remove both the short and the long sentences. There is also no noteworthy difference between the lengths of sentences.

From the results, we can conclude that our second assumption is true in at least the WT2g collection. However, the length of the queries could also have an effect. We conduct an additional experiment in Chapter 3.3.3 in order to see the effect of the query lengths on the placement of the query terms in sentences of the documents. We expect that the differences in Table 3.2 are greater in shorter queries than in longer queries. This is based on the assumption that shorter queries are comprised almost entirely of nouns whereas longer queries are more likely to contain other lexical classes. For example, a single query term is always a noun. Additional query terms can be from any of the lexical classes, such as semantically related nouns or descriptors such as adjectives or verbs. However, we leave the analysis of this assumption for a future study.

3.3.3 Query Length Analysis

In the topics 401-450, there are 3 queries with 1 query term, 25 queries with 2 query terms, and 22 queries with 3 query terms. We conduct the previous experiment and group the statistics by the length of the queries. The results are shown in Table 3.3 as follows:

M_i is the value of the measure M when the query length is i. $AvgD_1 > AvgD_2 > AvgD_3$ and $AvgP_1 < AvgP_2 < AvgP_3$ for the terms in both halves of sentences. This means that the difference in the placement of the terms are larger in shorter queries. Furthermore,

Query Length	Side	$AvgD$	$AvgP$	$AvgPN$	$AvgPL$	$AvgSL$
1	Left	1.0973	0.8513	0.915	0.8908	0.9775
	Right	1.084	0.8243	0.8757	0.8593	
2	Left	1.0312	0.9467	0.9687	0.9605	0.9944
	Right	1.0051	0.9871	0.9939	0.9914	
3	Left	1.0218	0.9676	0.969	0.968	0.9866
	Right	0.9894	1.0068	0.9929	0.9984	

TABLE 3.3: Results of the query length analysis

there is also a small difference in the length of sentences in shorter queries. However, the sample size is small and the difference is minuscule in longer queries. This shows that in at least the WT2g collection, the query term placement information is more effective at discriminating between the relevant and the non-relevant documents in shorter queries than in longer queries.

3.4 Effectiveness of Proposed Weighting Method

3.4.1 Design

In order to ensure that our proposed weighting method is effective, we conduct a third preliminary experiment. We partition sentences in a document into three parts, $\{p_1\ p_2\ p_3\}$, where $|p_1| = |p_3|$ and $|p_i|$ is the length of p_i. We then gives a score to each term t in sentences according to Equation 3.8:

$$Score(t) = \begin{cases} 1 & \text{if t } \in p_1 \cup p_3 \\ -1 & \text{if t } \in p_2 \end{cases} \tag{3.8}$$

The score of the term t in the document D is then:

$$Score(t, D) = \sum_{t_i \in D} Score(t_i) \tag{3.9}$$

where t_i is the ith occurrence of t in the document D. We can then conclude that Equation 3.10 is true in the majority of the cases. Equation 3.10 means that if $Score(t, D) > 0$, then t is more often found near the beginning or the end of sentences.

$$t \in \begin{cases} p_1 \cup p_3 & \text{if } Score(t, D) > 0 \\ p_2 & \text{if } Score(t, D) < 0 \end{cases} \tag{3.10}$$

We also obtain a balance of scores for the document D using Equation 3.11.

$$Balance(D) = \sum_{t \in D} Score(t, D) \tag{3.11}$$

We then adjust $|p_1|$ and $|p_3|$ until $Balance(D)$ is as close to 0 as possible for the document D. $Balance = 0$ means that the number of terms in p_2 is equal to the combined number of terms in p_1 and p_3. $Balance > 0$ means that the number of terms in p_2 is less than the combined number of terms in p_1 and p_3. The terms are stemmed. We will confirm that our propose weighting method is effective if the highest scoring terms are more important in the document than the lower scoring terms. The source code for the weighting method parser we use in this experiment is found in Appendix C.2.

3.4.2 Results

We conduct this experiment on the book *Hard Times* by Charles Dickens [67]. We further extract the nouns of out the terms in the document using the Stanford Log-linear Part-Of-Speech Tagger [68]. The ten highest and lowest scoring nouns are shown in Table 3.4 as follows:

Term	Score	Term	Score
mr.	86	head	-24
louisa	43	ladi	-17
mrs.	31	hand	-16
gradgrind	27	countri	-16
tom	23	time	-16
bounderbi	22	bank	-13
father	18	ey	-13
slackbridg	15	sparsit	-13
wai	15	boi	-13
miss	15	mean	-11

TABLE 3.4: Results of the weighting method preliminary experiment

$Balance = -117$ for this document. This means that there are 117 more words in p_2 than in both p_1 and p_3 combined. The results show that the highest scoring terms include "louisa", "gradgrind", and "bounderbi", which are the main characters of the book. "tom", "slackbridg", and "sparsit" are minor characters in the book. Since $Balance = -117$, we would expect that the above terms would have a negative score if the positions of the terms were random, which is not the case. The frequency of

the terms are not relevant since each occurrence of a term has a more or less equal (depending on *Balance*) chance to be in p_2 or $p_1 \cup p_3$. This means that in at least the book *Hard Times*, our proposed weighting method is able to effectively discriminate between terms that are important in the document and not important in the document. Additional results are found in Appendix C.3.

Chapter 4

Integration of Term Location into BM25

4.1 Design of the Reward Formula

4.1.1 Term Location

We assume that the most important terms in the document are near the beginning and the end of the sentences. This implies that the least important terms in the document are near the middle of the sentences. We can then measure the distance of a term from the middle of the sentence in order to determine its importance in the document, where a term that is more distant is more important. We set $q(t, D)$ to be the distance of the term t from the middle of the sentence in the document D as follows:

$$q(t, D) = |Mid(t, D) - Pos(t, D)| \tag{4.1}$$

$$\text{where } Mid(t, D) = \frac{SL(t, D) - 1}{2} \tag{4.2}$$

where SL(t,D) is the length of the sentence in a document D that contains t and $Pos(t, D)$ is the location of t in the sentence in D. We use the document-average of the distances of t from the middle of the sentences in D instead of the distinct distances.

We explain this design decision in Chapter 4.4. We define $r(t, D)$ as the document-average of the distances of t as follows:

$$r(t, D) = \frac{\sum_{t_i \in D} q(t_i, D)}{tf(t, D)} \tag{4.3}$$

where t_i is the ith occurrence of t in D and $tf(t, D)$ is the term frequency of t in D.

We use parameters to control the spread of the distribution based on the length of the sentence. The parameters are needed since the shape of the distribution that results in optimal IR performance is not known. We define $m(t, D)$ to be the document-average of the lengths of the sentences that contain t in D as follows:

$$m(t, D) = \frac{\sum_{t_i \in D} SL(t_i, D)}{\beta * tf(t, D)} + \gamma \tag{4.4}$$

where β and γ are tuning parameters for the spread of the distribution dependently and independently of the length of the sentence, respectively. The β parameter has a larger effect in longer sentences since its effect is proportional to the lengths of the sentences. The γ parameter has a proportionally smaller effect in longer sentences since its effect is the same in all sentences. Therefore both parameters are needed in order to have more control over the spread of the distribution in sentences of varying lengths. The effects of the parameters are illustrated in Chapter 4.1.3.

4.1.2 Kernel Functions

In order to measure the distance of the terms from the middle of the sentences, we fit a probability distribution or a kernel function over each sentence. We then adjust the weight of each term based on the value of the distribution at the location of the term. The value of the distribution is higher near the beginning and the end of the sentences. Since our preliminary experiments have shown that nouns and important terms in the documents are more often found near the beginning and the end of the sentences. The weight of the term is the probability that the term is a noun and important in the document. In this experiment, we use the following kernel functions:

$$\text{Gaussian - } Kernel(r, m) = 1 - e^{\frac{r^2}{-2m^2}} \tag{4.5}$$

$$\text{Uniform - } Kernel(r,m) = 0 \tag{4.6}$$

$$\text{Triangle - } Kernel(r,m) = \frac{r}{m} \tag{4.7}$$

$$\text{Cosine - } Kernel(r,m) = 1 - \frac{1+\cos\frac{r\pi}{m}}{2} \tag{4.8}$$

$$\text{Circle - } Kernel(r,m) = 1 - \sqrt{1-\left(\frac{r}{m}\right)^2} \tag{4.9}$$

$$\text{Quartic - } Kernel(r,m) = 1 - \left(1-\left(\frac{r}{m}\right)^2\right)^2 \tag{4.10}$$

$$\text{Epanechnikov - } Kernel(r,m) = \left(\frac{r}{m}\right)^2 \tag{4.11}$$

$$\text{Triweight - } Kernel(r,m) = 1 - \left(1-\left(\frac{r}{m}\right)^2\right)^3 \tag{4.12}$$

We supplement the kernel function with Equation 4.13. This ensures that we always give maximum reward to the terms that are adjacent or nearly adjacent to the beginning or the end of the sentences. This is parallel to our assumptions. The terms near the beginning and the end of the sentences are likely to be nouns and important in the document. The nearer the term is, the higher the likelihood. Therefore the terms that are adjacent or nearly adjacent to the beginning and the end of the sentences are almost certainly nouns and important in the document. Our preliminary experiment in Chapter 3.4 shows that this is true.

$$Reward(r,m) = \begin{cases} 1 & \text{if } r >= m \\ Kernel(r,m) & \text{if } r < m \end{cases} \tag{4.13}$$

The definition of nearly adjacent changes based on the β and γ parameters. Figure 4.1, 4.2, and 4.3 illustrates the shapes of the kernel functions we use in this experiment.

In Figures 4.1, 4.2, and 4.3, $r(t,D) >= m(t,D)$ for the terms under the red line. Maximum reward is given to those terms as per Equation 4.13. As $m(t,D)$ increases, the number of terms that are given maximum reward decreases. This is illustrated in Chapter 4.1.3. Evans and Zhai [30] described the phrase normalization problem in phrase-based indexing, where semantically different phrases are syntactically similar. Our kernel-based implementation is also affected by the phrase normalization problem. For example, the phrase "junior college" at the beginning of the sentence is equivalent

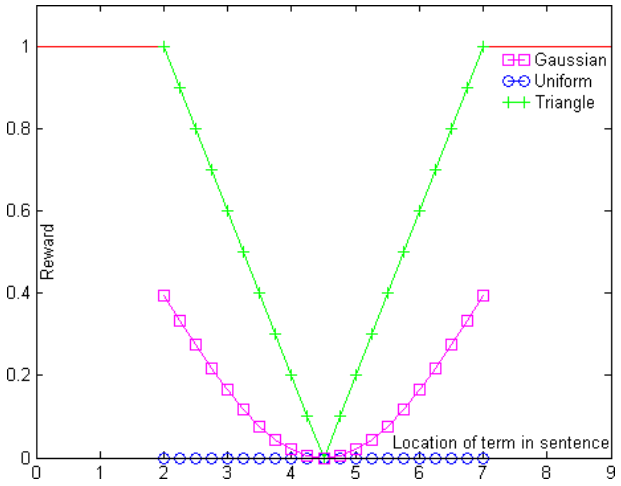

FIGURE 4.1: The Gaussian, Uniform, and Triangle kernel functions

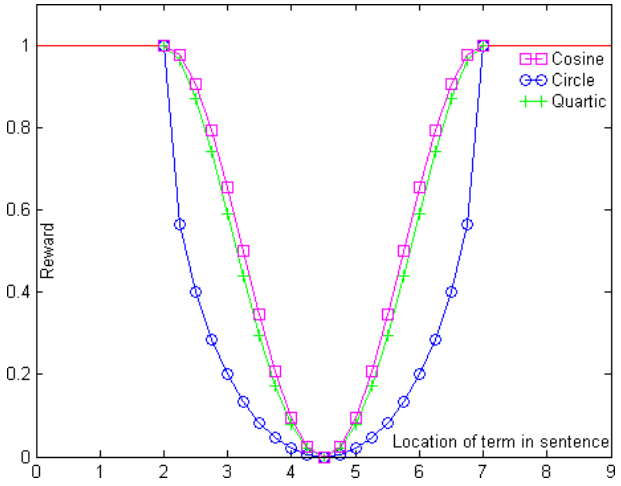

FIGURE 4.2: The Cosine, Circle, and Quartic kernel functions

to the phrase "college junior" at the end of the sentence. Therefore we plan on exploring non-symmetric kernel functions in the future as a possible solution, such as those in [50].

Similar to our preliminary experiments, we calculate the statistics for the terms that occur in the left half of the sentences and in the right half of the sentences separately. Therefore we use the document-average of the reward given to t in the left half of the sentences in D and in the right half of the sentences in D. We define the averaged reward (RA) as follows:

$$RA(t,D) = \begin{cases} \frac{Reward(r(t,D)_l, m(t,D)) + Reward(r(t,D)_r, m(t,D))}{2} & \text{if } r(t,D)_l > 0 \text{ and } r(t,D)_r > 0 \\ Reward(r(t,D)_l, m(t,D)) & \text{if } r(t,D)_l > 0 \text{ and } r(t,D)_r = 0 \\ Reward(r(t,D)_r, m(t,D)) & \text{if } r(t,D)_l = 0 \end{cases} \tag{4.14}$$

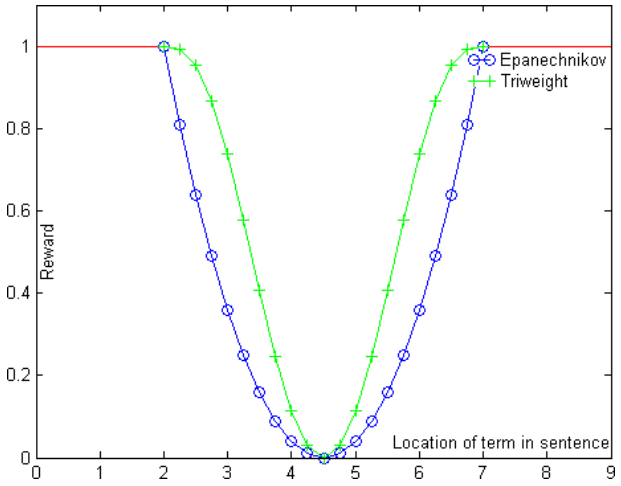

FIGURE 4.3: The Epanechnikov and Triweight kernel functions

where $r(t, D)_l$ is r for t in the left half of the sentences in D and $r(t, D)_r$ is r for t in the right half of the sentences in D.

4.1.3 Effect of the β and γ Parameters

Figure 4.4 and 4.5 illustrates the effect of changing the β parameter on the reward given to the terms in sentences of lengths 10 and 20, respectively. Since β is dependent on the lengths of the sentences, its effect is proportionally similar in sentences of varying lengths. This can be seen in the figures. If β is equal, the width of the distributions are similar in sentences of length 10 and length 20. β has a larger effect in longer sentences.

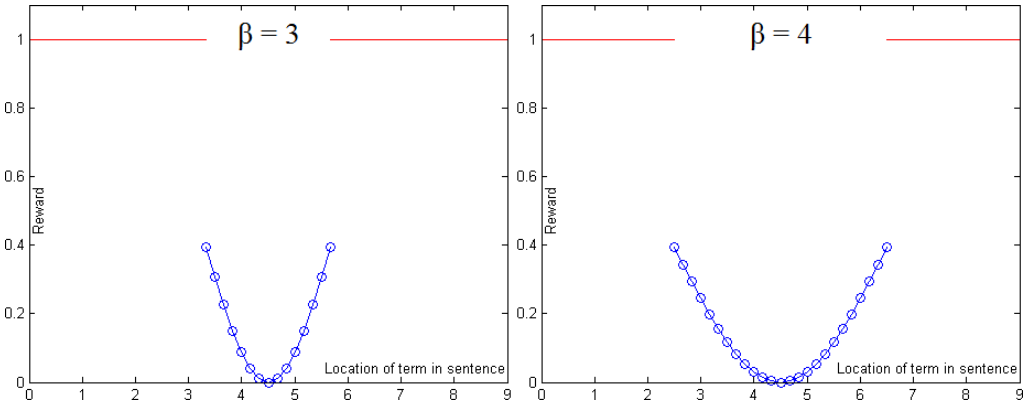

FIGURE 4.4: Effect of the β parameter in sentences of length 10

Figure 4.6 and 4.7 illustrates the effect of changing the γ parameter on the reward given to the terms in sentences of lengths 10 and 20, respectively. Since γ is independent of the lengths of the sentences, its effect is always constant and is proportionally different

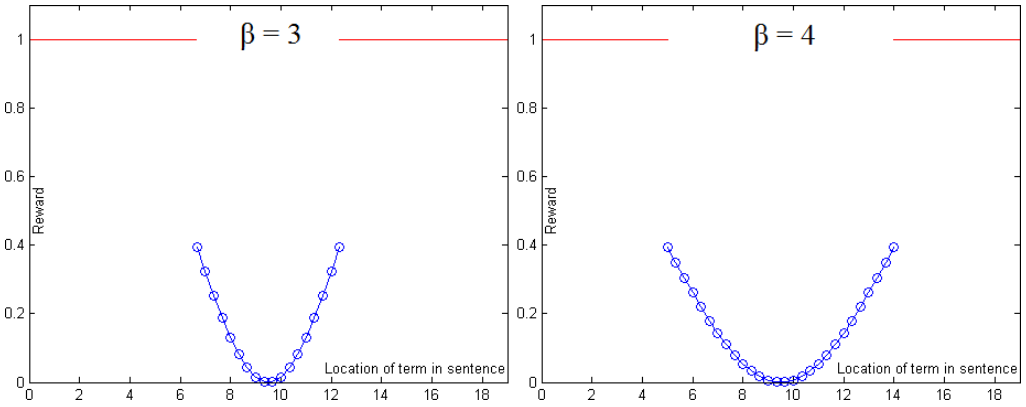

FIGURE 4.5: Effect of the β parameter in sentences of length 20

in sentences of varying lengths. This can be seen in the figures. When $\gamma = 2$ is equal, the width of the distribution is much wider in sentences of length 10 than sentences of length 20. γ has a proportionally larger effect in shorter sentences.

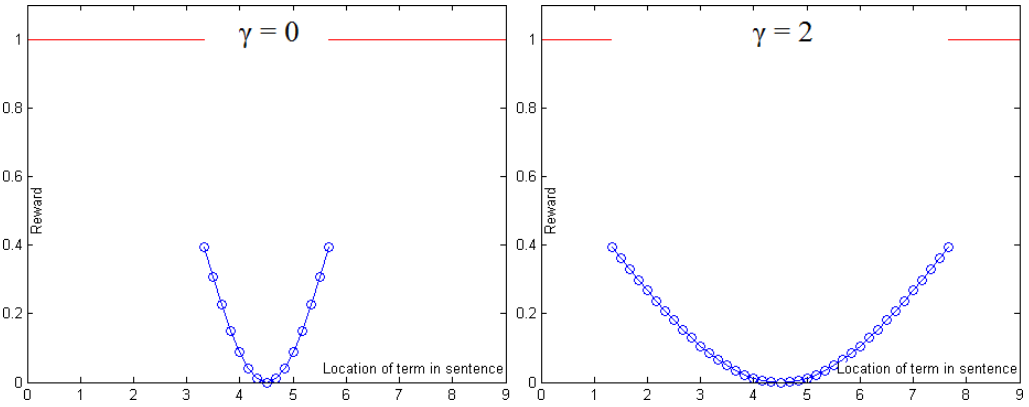

FIGURE 4.6: Effect of the γ parameter in sentences of length 10

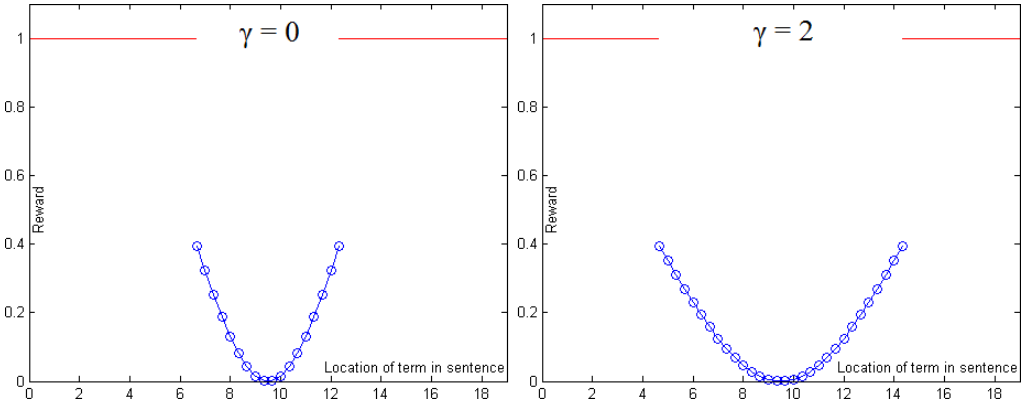

FIGURE 4.7: Effect of the γ parameter in sentences of length 20

4.1.4 Sentence Length Normalization

We normalize the reward given to a term according to the length of the sentence that it is in. We found that there is no noteworthy difference in the lengths of the sentences in the relevant and the non-relevant documents in our query length analysis preliminary experiment in Chapter 3.3.3. However, it is statistically more likely for a term to be nearer to the beginning or the end of the sentence in shorter even-length sentences. This is due to how we calculate the distance of the term from the middle of the sentence in Equation 4.1. For example, if we have the terms t_1 and t_2 in the sentence s_1 where $|s_1| = 4$ and $p(t_i) = i - 1$, then $q(t_i, D) = |1.5 - i|$, The average distance of the terms is then 1 which is $33.\overline{3}\%$ nearer to the beginning or the end of s_1 than in reality. This effect is greatly diminished in longer sentences. Therefore we counteract it by removing the sentences s from the collection where $|s| < 7$. We also normalize the Equation 4.13 using the log normalized lengths of the sentences, where more reward is given for the terms in longer sentences. We defined the normalized reward (RN) as follows:

$$RN(t, D) = \frac{log_2(1 + AvgSL(t, D))}{log_2(1 + AvgSL)} * RA(r(t, D), m(t, D)) \tag{4.15}$$

where $AvgSL$ is the average of the lengths of the sentences in the collection and $AvgSL(t, D)$ is the average length of the sentences that contains t in D.

4.2 Merge into BM25

In this experiment, we use the BM25 weighting model as our base weighting model. BM25 is defined as follows:

$$Score(t, D) = TF(t, D) * IDF(t) \tag{4.16}$$

$$\text{where } TF(t, D) = \frac{(k_3 + 1) * tf(t, D) * qtf(t)}{(k_3 + qtf(t)) * K} \tag{4.17}$$

$$K = k_1 * \left(1 - b + \frac{b * |D|}{AvgDL}\right) + tf(t, D) \tag{4.18}$$

$$\text{and } IDF(t) = log_2 \frac{N - n(t) + 0.5}{n(t) + 0.5} \tag{4.19}$$

where k_1, k_3, and b are tuning parameters for BM25. $qtf(t)$ is the frequency of t in the query, $|D|$ is the number of terms in D, and $AvgDL$ is the average length of the documents in the collection. N is the number of documents in the collection and $n(t)$ is the number of documents in the collection that contain t. We modify $TF(t, D)$ to account for the reward given to the terms based on the location of the terms in the sentences. We define the first Term Location score (TL1) as follows:

$$TL1(t, D) = \frac{(k_3 + 1) * RN(t, D) * tf(t, D) * qtf(t)}{(k_3 + qtf(t)) * K_{TL}} \tag{4.20}$$

$$\text{and } K_{TL} = k_1 * \left(1 - b + \frac{b * |D|}{AvgDL}\right) + RN(t, D) * tf(t, D) \tag{4.21}$$

4.3 Query Length Normalization

We normalize the reward given a term according to the length of the query. This is due to the findings in our query length analysis preliminary experiment in Chapter 3.3.3. In at least the WT2g collection, the query term placement information is better at discriminating between the relevant and the non-relevant documents in shorter queries. Therefore we want to be more stringent in rewarding terms when the query is longer. We propose the Query Length Score (QLS) to accomplish this as follows:

$$QLS = \left(\frac{0.5}{0.5 + |Q|}\right)^{\frac{2}{3}} \tag{4.22}$$

where $|Q|$ is the number of terms in the query Q. The shape of QLS is illustrated in Figure 4.8. It is function that decreases slowly as the length of the query is increased. The constants in QLS were chosen arbitrarily and have proven to be effective in our experiments. We leave the analysis of QLS for a future study.

We integrate QLS with $TL1$ to form the second Term Location score (TL2) as follows:

$$TL2(t, D) = QLS * TL1(t, D) + (1 - QLS) \tag{4.23}$$

We then integrate our model into BM25 using linear combination to form the Term Location Score (TEL) as follows:

$$TEL(t, D) = ((1 - \alpha) * TF(t, D) + \alpha * TL2(t, D)) * IDF(t) \tag{4.24}$$

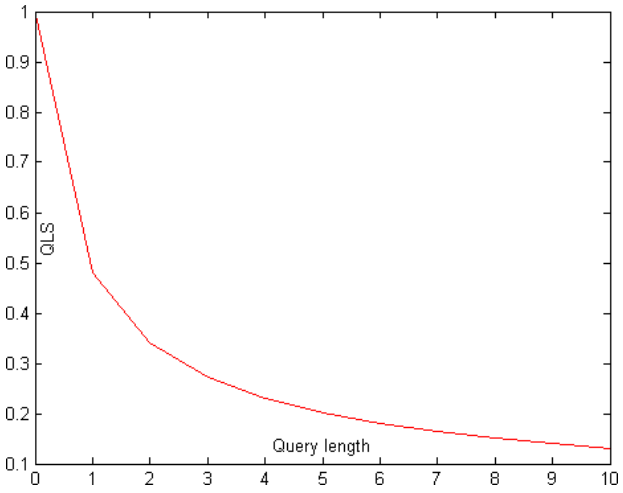

FIGURE 4.8: Query Length Score (QLS)

where α is the tuning parameter for the linear combination. α controls the contribution of our model in the score of the documents, where a higher α means that our model contributes more. The full source code of our implementation is found in Appendix D.

4.4 Design Decisions

The values q and SL were aggregated into the values r and m in Equation 4.3 and 4.4, respectively. Using distinct the values of q and SL proved to be too demanding on the systems we use to perform our experiment, and drastically increased query running time due to the increase of the I/O required. The values r and m were calculated during indexing and therefore have no impact on query running time. Along with additional optimization average running time for one iteration of retrieval on the WT2g data set improved by approximately 875%, from approximately 70 seconds to approximately 8 seconds. Tao and Zhai [23] used other measures such as Min and Max to model the proximity between terms. We will investigate similar methods for aggregating the values of q and SL in the future.

We did not take into account the location of the terms in the queries because we assume that the queries are short and will rarely compose a complete sentence. Our model is therefore not suitable for the queries. We also assume that all terms in a query are important and contribute equally to the topics sought by the query. Therefore it is also not necessary to apply our model to the terms in the queries.

Chapter 5

Experimental Settings

5.1 Collections

We conduct our experiments on the five standard TREC collections shown in Table 5.1. These collections vary in both size and content and are representative.

Collection Name	# of Docs	Topics
WT2g	247,491	401-450
Disk4&5	528,155	301-450
WT10g	1,692,096	451-550
Blogs06	3,215,171	851-950
GOV2	25,178,548	701-850

TABLE 5.1: Collections we use our experiments

WT2g is a 2 GB size crawl of general web documents. It was used in the TREC 1999 Web track. Disk4&5 is comprised of news-wire articles from sources such as the Financial Times (FT) and the Federal Register (FR) and is usually considered as high quality text data with little noise. Disk4&5 was used in the TREC 1997-1999 Ad hoc tasks. WT10g is a 10 GB size crawl of general web documents. It was used in the TREC 2000 and 2001 Web tracks. Blogs06 is a 148 GB size crawl of feeds from late 2005 to early 2006 with associated permalink and homepage documents. It was used in the TREC 2006-2008 Blog tracks. As recommended in [69], we only use the permalink documents. GOV2 is a 426 GB size crawl of .gov sites. It was used in the TREC 2004-2006 Terabyte tracks.

5.2 Evaluation Metrics

Precision P in IR is the proportion of retrieved documents that are relevant and is measured with the following formula:

$$P = \frac{|R \cap T|}{|T|} \tag{5.1}$$

where R is the set of documents that are relevant and T is the set of documents that are retrieved. Given a set A, $|A|$ is the number of items in the set A. Precision of the top k documents is then defined as $P@(k)$ as follows:

$$P@(k) = \frac{|R \cap (T_1 \cup T_2 \ldots T_k)|}{k} \tag{5.2}$$

where T_i is the ith document in T.

The mean average precision (MAP) [70] is the mean of the average of the precision of each query and is defined as follows:

$$MAP = \frac{\sum_{q \in Q} Avg.P(q)}{|Q|} \tag{5.3}$$

$$\text{where } Avg.P(q) = \frac{\sum_{k=1}^{n} (P@(k) * Rel(k))}{|R|} \tag{5.4}$$

$$\text{and } Rel(k) = \begin{cases} 1 & \text{if } T_k \in R \\ 0 & \text{if } T_k \notin R \end{cases} \tag{5.5}$$

where Q is the set of queries and q is a query in Q. In our experiment, $n = 1000$ in Equation 5.4.

MAP is the standard measure of overall retrieval performance in IR. $P@(k)$ is a standard measure of retrieval precision. We also want to emphasize the precision of the top documents retrieved. In search, it is often the case that only the top documents are viewed while the rest of the documents are discarded. Spink et al found that 28.6% [71] and 58% [72] of their users only viewed one page of the results. In a separate study, Silverstein et al [73] found that 85.2% of their users only viewed one page of the results. Therefore we use the $P@(k)$ measure for $k = \{5\ 10\ 20\}$ to emphasize the significance of the top k documents. We use the Wilcoxon signed-rank test with $p < 0.05$ to test for statistical significance since the results are paired.

5.3 Terrier Settings

The Terrier properties used for each collection are the recommended properties from the Terrier group. The exact properties can be found in Appendix E. These properties are used when generating the primary index and running retrieval using Terrier.

5.4 Baselines

We compare the performance of our model against the following weighting models:

1. BM25, with $k_1 = 1.2$ and $k_3 = 8$. We adjust b in the range of [0.1, 0.9] in steps of 0.1. We run one pass of retrieval for each value of b. We order the results on MAP in descending order and use the value of b with the best result for our baseline. This process is repeated for each collection.

2. DirichletLM. We adjust μ in the range of [100, 3000] in steps of 100. We find the optimal value of μ for each collection using the above process.

Table 5.2 and 5.3 shows the optimal parameters and the baseline performance of BM25 and DirichletLM, respectively.

Collection	b	MAP	P@5	P@10	P@20
WT2g	0.2	0.3167	0.512	0.468	0.387
disk4+5	0.3	0.2176	0.468	0.4293	0.3613
WT10g	0.3	0.2134	0.3918	0.3276	0.2776
Blogs06	0.2	0.3195	0.638	0.641	0.6095
Gov2	0.4	0.3008	0.6094	0.5779	0.5406

TABLE 5.2: BM25 Baseline Parameters and Performance

Collection	μ	MAP	P@5	P@10	P@20
WT2g	1500	0.3059	0.508	0.454	0.387
disk4+5	500	0.219	0.456	0.4167	0.3627
WT10g	600	0.2168	0.3531	0.3173	0.2745
Blogs06	1700	0.3125	0.608	0.613	0.5935
Gov2	800	0.2983	0.5919	0.551	0.5272

TABLE 5.3: DirichletLM Baseline Parameters and Performance

5.5 System Settings

We conduct our experiment using the Terrier 3.5 [74] IR platform. Terrier is an open source IR platform developed in Java at the University of Glasgow. We use the Linux Mint 15 and 16 x64 and Ubuntu 12.04 x86 operating systems with Java SE 8u5 x64 and x86, respectively.

5.6 Secondary Index Settings

The experimental settings in which we index our collections is similar to the experimental settings in our preliminary experiments. We remove standard English stop-words from the documents and stem the remaining terms using an English Porter stemmer [18]. In the documents in the WT2g, WT10g, Blogs06, and Gov2 collections, we ignore the headers by only processing the text between the </DOCHDR> and </DOC> tags. In the documents in the disk4+5 collection, we only process the text between the <TEXT> and </TEXT> tags. We remove all HTML tags, numbers, and characters that the Stanford Tagger cannot parse. We use the English Stanford Tagger 3.3.1 [68] for part-of-speech tagging and sentence identification. We add newlines to the Stanford Tagger's default set of sentence delimiters. Punctuation that are tagged as any of {# $ " ( ) , : “} are removed prior to processing. We also remove all HTML tags, numbers, and characters that the Stanford Tagger cannot parse from the documents. We only keep characters with Unicode numbers in the ranges of [20, 2F] and [3A, 7F].

We only keep a sentence s if $|s| \geq 7$ and $|s| \leq 20$ where $|s|$ is the number of terms in s. We calculate the statistics for the terms that occur in the left half of sentences and in the right half of sentences separately. A term t in the document D is in the left half of the sentence if $Mid(t, D) - p(t, D) < 1$. t in D is in the right half of the sentence if $Mid(t, D) - p(t, D) > 1$. t in D where $Mid(t, D) - p(t, D) = 0$ are ignored.

We store the term location information, q and SL, in a h2 1.3.174 database. The source code for the term location indexer is found in Appendix B.2. We do not store the term location information for all the terms in a document, only the corresponding query terms. This is done in consideration of running time and disk space. This has no effect

on the performance of our model as long as the baseline parameters of BM25 for each collection are not changed.

5.7 Retrieval and Evaluation

We remove standard English stop-words from the queries and stem the remaining terms using an English Porter stemmer [18]. The length of the query is the number of terms in the query after stop-word removal. In this experiment, we set $\alpha = 0.2$, $\beta = 3$, and $\gamma = 3$. We believe that our model is robust enough to not require parameter optimization. We set $k_1 = 1.2$, $k_3 = 8$, and b to the optimal value from our baseline parameter optimization process shown in Table 5.2. We set $AvgSL = 10.5$ in Equation 4.15 for every collection based on the results from the noun placement preliminary experiment in Chapter 3.2.

We run the first pass of retrieval with BM25. We sort the documents on score in descending order. We run a second pass of retrieval on the top 1000 documents using our model. For each document, the score from the first pass of retrieval is multiplied by $(1 - \alpha)$ and the score from the second pass of retrieval is multiplied by α. This implementation is equivalent to Equation 4.24. The source code of this algorithm is found in Appendix D.2. We use trec_eval 9 to evaluate the MAP and the $P@(k)$ measures for the retrieval results. We use the function WilcoxonSignedRankTest.wilcoxonSignedRankTest() from the Apache Commons Math 3.2 API to test for statistical significance.

Chapter 6

Experimental Results

6.1 Effectiveness of Our Model

6.1.1 WT2g

The experimental results in Table 6.1 are with $k_1 = 1.2$, $k_3 = 8$, $b = 0.2$, $\alpha = 0.2$, $\beta = 3$, and $\gamma = 3$.

The results show that the performance of our model is statistically significantly better than BM25 in MAP, P@5, P@10, and P@20 and LM in MAP, P@5, and P@10. The Uniform kernel function do not significantly improve BM25 and is the worst performing kernel function in all metrics. The Quartic kernel function is significantly better than BM25 in all metrics. However the performance improvement is small compared to the Gaussian and the Circle kernel functions. The Gaussian and the Circle kernel functions is significantly better than BM25 in MAP, P@5, and P@10 and significantly better than LM in MAP, and P@5. The Gaussian the the Circle kernel functions are the best performing kernel functions overall. All kernel functions except for the Uniform and the Triweight kernel functions perform better than BM25 and LM in every metric.

The MAP and P@20 performance of all of the kernel functions are better than the baseline weighting models. However, only some of the improvements are statistically significant. There are large differences between the best and the worst performances in P@5 and P@10. The differences between the highest % performance improvement over BM25 in P@5 and P@10 and the lowest are 4.69% and 5.12% (1.71% without

Baseline Model	MAP	P@5	P@10	P@20
BM25	0.3167	0.512	0.468	0.387
LM	0.3059	0.508	0.454	0.387
Kernel	MAP	P@5	P@10	P@20
Gaussian	0.3223*+ 1.77%, 5.36%	0.52* 1.56%, 2.36%	**0.482*** 2.99%, 6.17%	**0.396** 2.33%, 2.33%
Uniform	0.3226 1.86%, 5.46%	0.524+ 2.34%, 3.15%	0.456+ -2.56%, 0.44%	**0.396** 2.33%, 2.33%
Triangle	0.3179 0.38%, 3.92%	0.504 -1.56%, -0.79%	0.472* 0.85%, 3.96%	0.389* 0.52%, 0.52%
Circle	**0.3235***+ 2.15%, 5.75%	**0.528*** 3.13%, 3.94%	0.48* 2.56%, 5.73%	0.395 2.07%, 2.07%
Cosine	0.3186 0.6%, 4.15%	0.516* 0.78%, 1.57%	0.476* 1.71%, 4.85%	0.389* 0.52%, 0.52%
Quartic	0.3199*+ 1.01%, 4.58%	0.516* 0.78%, 1.57%	0.474* 1.28%, 4.41%	0.39* 0.78%, 0.78%
Epanechnikov	0.3201*+ 1.07%, 4.64%	0.52* 1.56%, 2.36%	0.48 2.56%, 5.73%	0.393 1.55%, 1.55%
Triweight	0.3179 0.38%, 3.92%	0.508 -0.78%, 0%	0.476* 1.71%, 4.85%	0.388* 0.26%, 0.26%

TABLE 6.1: Results for WT2g, * and + denotes statistical significance over BM25 and LM, respectively. The best result for each metric is in bold. The percentages below each value is the % improvement over BM25 and LM, respectively

the Uniform kernel function), respectively. P@5 and P@10 are the most volatile out of the metrics that we use in this experiment. However, the differences are still large but only in P@5 if the Uniform kernel function is treated as an outlier and ignored. This can indicate that the performance of our model in the WT2g collection is highly dependent on the values of the parameters or the kernel function used. This is because the different kernel functions can have vastly differently optimal values of α, β, and γ. We will analyze this further in our robustness study in Chapter 6.2.

We choose two kernel functions that perform well across all collections, Gaussian and Circle, to represent our model. We illustrate the % performance difference between those kernel functions and the baseline weighting models in Figure 6.1.

The figure shows that the performance of our model is consistently better than the baseline weighting models. The % performance improvement is also similar in every metric. This means that the performance of our model is fairly consistent in at least the top 20 documents. The performance of the Gaussian and the Circle kernel functions are almost equal overall.

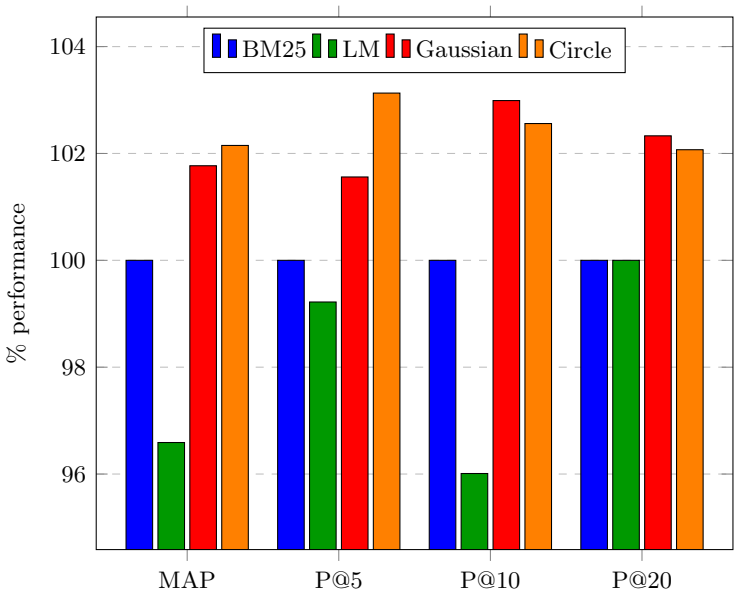

FIGURE 6.1: WT2g, % performance improvement

6.1.2 disk4+5

The experimental results in Table 6.2 are with $k_1 = 1.2$, $k_3 = 8$, $b = 0.3$, $\alpha = 0.2$, $\beta = 3$, and $\gamma = 3$.

Baseline Model	MAP	P@5	P@10	P@20
BM25	0.2176	**0.468**	**0.4293**	0.3613
LM	0.219	0.456	0.4167	0.3627
Kernel	MAP	P@5	P@10	P@20
Gaussian	0.2201 1.15%, 0.5%	0.4627^{+} -1.13%, 1.47%	0.4247 -1.07%, 1.92%	0.3653* 1.11%, 0.72%
Uniform	0.2206 1.38%, 0.73%	0.4573^{+} -2.29%, 0.29%	0.4227 -1.54%, 1.44%	0.365* 1.02%, 0.63%
Triangle	**0.2209*** 1.52%, 0.87%	0.4613^{+} -1.43%, 1.16%	0.4267 -0.61%, 2.4%	0.364^{*+} 0.75%, 0.36%
Circle	0.2201 1.15%, 0.5%	0.4627^{+} -1.13%, 1.47%	0.4247 -1.07%, 1.92%	0.3647* 0.94%, 0.55%
Cosine	**0.2209*** 1.52%, 0.87%	0.4613^{+} -1.43%, 1.16%	0.4287 -0.14%, 2.88%	0.3643^{*+} 0.83%, 0.44%
Quartic	**0.2209*** 1.52%, 0.87%	0.4613^{+} -1.43%, 1.16%	0.428 -0.3%, 2.71%	0.3643^{*+} 0.83%, 0.44%
Epanechnikov	0.2206 1.38%, 0.73%	0.464^{+} -0.85%, 1.75%	0.4247 -1.07%, 1.92%	**0.366*** 1.3%, 0.91%
Triweight	0.2205* 1.33%, 0.68%	0.4613^{+} -1.43%, 1.16%	0.4287 -0.14%, 2.88%	0.364^{*+} 0.75%, 0.36%

TABLE 6.2: Results for disk4+5, * and $^{+}$ denotes statistical significance over BM25 and LM, respectively. The best result for each metric is in bold. The percentages below each value is the % improvement over BM25 and LM, respectively

The results show that the performance of our model is statistically significantly better than BM25 in MAP and P@20 and significantly better than LM in P@5 and P@20. However, the performance of our model is consistently worse than BM25 in P@5 and P@10. The Triangle, the Cosine, and the Quartic kernel functions are the best performing kernel functions overall. Overall the results are mixed. However, the performance of all of the kernel functions are moderately better than BM25 in MAP and P@20 and better than LM in all metrics. However, not all of the improvements are statistically significant.

There are no large differences between the best and the worst performances in any metric if the Uniform kernel function is treated as an outlier and ignored. This can indicate that our model is fairly robust in the disk4+5 collection. The performance of our model is only moderately better than BM25 in MAP and P@20 and worse than BM25 in P@5 and P@10. This can indicate that our model does not perform well in the top 10 documents in this data set but perform much better in documents 11-20. We analyze this further in our robustness study in Chapter 6.2.

We choose two kernel functions that perform well across all collections, Gaussian and Circle, to represent our model, although the Gaussian and the Circle kernel functions are not the best performing kernel functions in this data set. We illustrate the % performance difference between those kernel functions and the baseline weighting models in Figure 6.2.

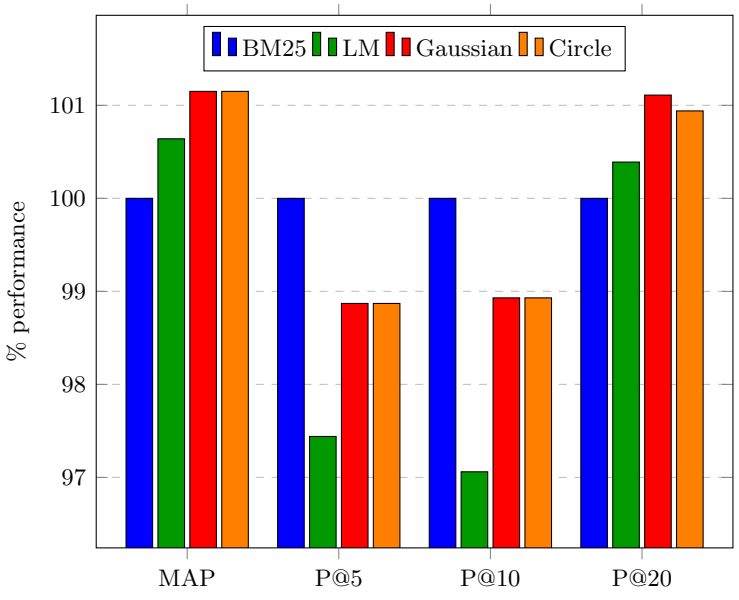

FIGURE 6.2: disk4+5, % performance improvement

The figure shows that the performance of our model is better than BM25 in MAP and P@20, but worse in P@5 and P@10. The performance of our model is consistently better than LM. The performance of both the Gaussian and the Circle kernel functions is around the same.

6.1.3 WT10g

The experimental results in Table 6.3 are with $k_1 = 1.2$, $k_3 = 8$, $b = 0.3$, $\alpha = 0.2$, $\beta = 3$, and $\gamma = 3$.

Baseline Model	MAP	P@5	P@10	P@20
BM25	0.2134	0.3918	**0.3276**	**0.2776**
LM	0.2168	0.3531	0.3173	0.2745
Kernel	MAP	P@5	P@10	P@20
Gaussian	0.2202* 3.19%, 1.57%	0.3898^{+} -0.51%, 10.39%	0.3255^{+} -0.64%, 2.58%	0.2765 -0.4%, 0.73%
Uniform	0.208 -2.53%, -4.06%	0.3816 -2.6%, 8.07%	0.3194^{+} -2.5%, 0.66%	0.2663 -4.07%, -2.99%
Triangle	0.2196* 2.91%, 1.29%	0.3898^{+} -0.51%, 10.39%	0.3265^{+} -0.34%, 2.9%	0.2755^{+} -0.76%, 0.36%
Circle	0.2202* 3.19%, 1.57%	**0.4*** 2.09%, 13.28%	0.3235^{+} -1.25%, 1.95%	0.277 -0.22%, 0.91%
Cosine	0.2197* 2.95%, 1.34%	0.3898^{+} -0.51%, 10.39%	0.3265^{+} -0.34%, 2.9%	0.277^{+} -0.22%, 0.91%
Quartic	0.2199* 3.05%, 1.43%	0.3918^{+} 0%, 10.96%	0.3255^{+} -0.64%, 2.58%	0.2755 -0.76%, 0.36%
Epanechnikov	**0.2206*** 3.37%, 1.75%	0.3918^{+} 0%, 10.96%	0.3224^{+} -1.59%, 1.61%	0.276 -0.58%, 0.55%
Triweight	0.2198* 3%, 1.38%	0.3898^{+} -0.51%, 10.39%	0.3265^{+} -0.34%, 2.9%	0.2765^{+} -0.4%, 0.73%

TABLE 6.3: Results for WT10g, * and $^{+}$ denotes statistical significance over BM25 and LM, respectively. The best result for each metric is in bold. The percentages below each value is the % improvement over BM25 and LM, respectively

The results show that the performance of our model is statistically significantly better than BM25 in MAP and P@5 and also statistically significantly better than LM in P@5, P@10, and P@20. However, the performance of our model is consistently worse than BM25 in P@10 and P@20. In P@5, only the Circle kernel function performs better than BM25. The performance of the Uniform kernel function is the worst overall and is worse than BM25 in every metric. The performance of the Circle kernel function is the best overall, similar to the WT2g collection. Overall the results are mixed. The fact

that there's a large improvement in MAP over BM25 for all kernel functions can mean that our model is not well suited for the top 20 documents in the WT10g data set.

We treat the Uniform kernel function as an outlier and ignore it. The remaining kernel functions significantly improve BM25 in MAP greatly. All of the kernel functions improve LM in every metric, though not all improvements are statistically significant. There are also no large differences between the best and the worst performances in P@10 and P@20. There is a large difference between the highest % improvement over BM25 in P@5 (2.09%) and the lowest (-0.51%). P@5 is the most volatile out of the metrics that we use in this experiment. The P@5 performance of the Circle kernel function can be an outlier as well. We analyze this further in our robustness study in Chapter 6.2.

We choose two kernel functions that perform well across all collections, Gaussian and Circle, to represent our model. We illustrate the % performance difference between those kernel functions and the baseline weighting models in Figure 6.3.

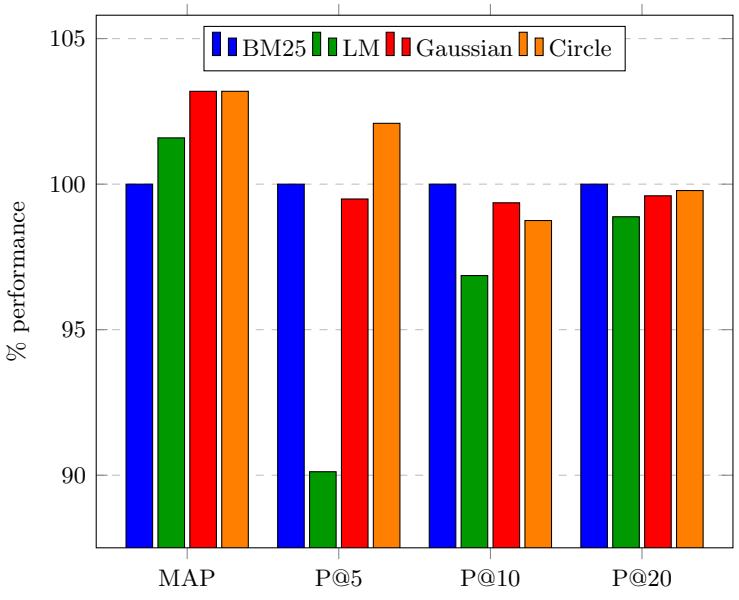

FIGURE 6.3: WT10g, % performance improvement

The figure shows that the performance of our model is better than BM25 in MAP and P@5, but worse than BM25 in P@10 and P@20. The performance of our model is better than BM25 for documents 1 to 5 and 11 to 20. However, the performance of our model is moderately worse than BM25 for documents 6 to 10, which affects the P@10 and P@20 performance of our model. The performance of the Circle kernel function is better than the performance of the Gaussian kernel function overall. However, it is slightly worse than the Gaussian kernel function for documents 6 to 10.

6.1.4 Blogs06

The experimental results in Table 6.4 are with $k_1 = 1.2$, $k_3 = 8$, $b = 0.2$, $\alpha = 0.2$, $\beta = 3$, and $\gamma = 3$.

Baseline Model	MAP	P@5	P@10	P@20
BM25	0.3195	0.638	0.641	0.6095
LM	0.3125	0.608	0.613	0.5935
Kernel	MAP	P@5	P@10	P@20
Gaussian	0.3238^{+} 1.35%, 3.62%	0.664* 4.08%, 9.21%	0.656* 2.34%, 7.01%	0.618 1.39%, 4.13%
Uniform	0.3158 -1.16%, 1.06%	0.63^{+} -1.25%, 3.62%	0.616 -3.9%, 0.49%	0.594 -2.54%, 0.08%
Triangle	**0.3241*+** 1.44%, 3.71%	**0.68*** 6.58%, 11.84%	0.652* 1.72%, 6.36%	**0.6225** 2.13%, 4.89%
Circle	0.3238^{+} 1.35%, 3.62%	0.668* 4.7%, 9.87%	**0.657*** 2.5%, 7.18%	0.6185 1.48%, 4.21%
Cosine	0.3239^{*+} 1.38%, 3.65%	0.676* 5.96%, 11.18%	0.653* 1.87%, 6.53%	0.621 1.89%, 4.63%
Quartic	0.3239^{*+} 1.38%, 3.65%	0.676* 5.96%, 11.18%	0.653* 1.87%, 6.53%	0.622 2.05%, 4.8%
Epanechnikov	0.3239^{+} 1.38%, 3.65%	0.674* 5.64%, 10.86%	0.656* 2.34%, 7.01%	0.6195 1.64%, 4.38%
Triweight	0.3239^{*+} 1.38%, 3.65%	0.676* 5.96%, 11.18%	0.651* 1.56%, 6.2%	0.622 2.05%, 4.8%

TABLE 6.4: Results for Blogs06, * and $^{+}$ denotes statistical significance over BM25 and LM, respectively. The best result for each metric is in bold. The percentages below each value is the % improvement over BM25 and LM, respectively

The results show that the performance of our model is statistically significantly better than BM25 in MAP, P@5, and P@10 and significantly better than LM in MAP and P@5. The P@20 performance of our model is better than BM25 but the improvement is not significant. The P@10 and P@20 performance of our model is better than LM but likewise, the improvement is not significant. The Uniform kernel function is the worst performing overall, performing worse than BM25 in all metrics. The improvement in P@5 is large while the improvement in MAP, P@10, and P@20 is moderate.

We treat the Uniform kernel function as an outlier and ignore it. Overall the performance of our model is significantly better than BM25 in every metric except for P@20, where it is at least equivalent. There are no large differences between the best and the worst performances in MAP, P@10, and P@20. There is a large difference between the highest % improvement over BM25 in P@5 (6.58%) and the lowest (4.08%). Similar

to the WT2g and the WT10g collections, this can be due to the volatility of the $P@5$ metric. We analyze this further in our robustness study in Chapter 6.2.

We choose two kernel functions that perform well across all collections, Gaussian and Circle, to represent our model. We illustrate the % performance difference between those kernel functions and the baseline weighting models in Figure 6.4.

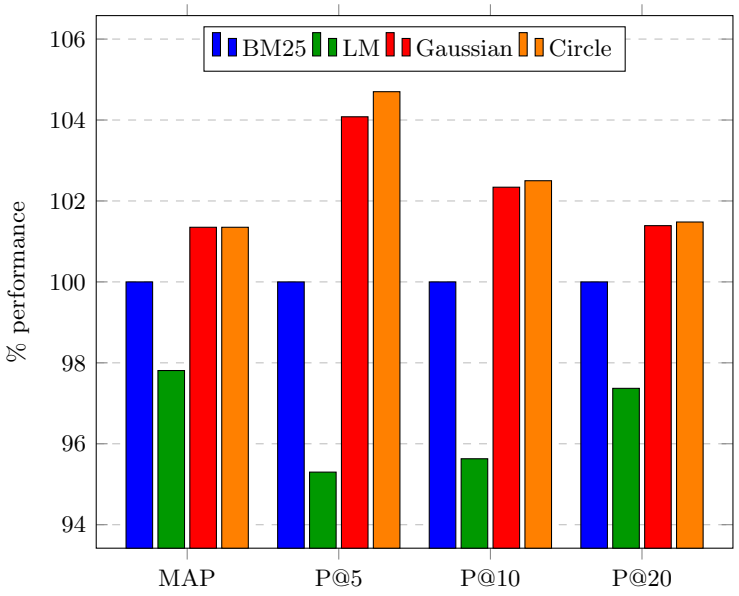

FIGURE 6.4: Blogs06, % performance improvement

The figure shows that the performance of our model is better than the baseline weighting models in every metric. The performance of our model is much better than BM25 and LM for documents 1 to 5 and moderately better than BM25 for documents 11 to 20. The performance of the two kernel functions are similar.

6.1.5 Gov2

The experimental results in Table 6.5 are with $k_1 = 1.2$, $k_3 = 8$, $b = 0.4$, $\alpha = 0.2$, $\beta = 3$, and $\gamma = 3$.

The results show that the performance of our model is statistically significantly better than BM25 in MAP, $P@5$, and $P@10$ and significantly better than LM in MAP and $P@5$. The performance of our model is consistently worse than BM25 in $P@20$. All of the kernel functions except for the Uniform kernel function significantly improves BM25 in MAP and $P@10$. The Uniform kernel function performs the worst overall and performs worse than BM25 in every metric and performs worse than LM in every metric

Baseline Model	MAP	P@5	P@10	P@20
BM25	0.3008	0.6094	0.5779	**0.5406**
LM	0.2983	0.5919	0.551	0.5272
Kernel	MAP	P@5	P@10	P@20
Gaussian	**0.3045*+** 1.23%, 2.08%	**0.6174*+** 1.31%, 4.31%	**0.5913*** 2.32%, 7.31%	0.5383 -0.43%, 2.11%
Uniform	0.2961 -1.56%, -0.74%	0.5906 -3.09%, -0.22%	0.5617 -2.8%, 1.94%	0.5191 -3.98%, -1.54%
Triangle	0.3041*+ 1.1%, 1.94%	0.6067+ -0.44%, 2.5%	0.5805* 0.45%, 5.35%	0.5356 -0.92%, 1.59%
Circle	**0.3045*+** 1.23%, 2.08%	0.6148*+ 0.89%, 3.87%	0.5886* 1.85%, 6.82%	0.5376 -0.55%, 1.97%
Cosine	0.3043*+ 1.16%, 2.01%	0.6054+ -0.66%, 2.28%	0.5812* 0.57%, 5.48%	0.5383 -0.43%, 2.11%
Quartic	0.3043*+ 1.16%, 2.01%	0.604+ -0.89%, 2.04%	0.5812* 0.57%, 5.48%	0.5393 -0.24%, 2.3%
Epanechnikov	0.3044*+ 1.2%, 2.04%	0.6107*+ 0.21%, 3.18%	0.5859* 1.38%, 6.33%	0.5383 -0.43%, 2.11%
Triweight	0.3043*+ 1.16%, 2.01%	0.6054+ -0.66%, 2.28%	0.5805* 0.45%, 5.35%	0.5379 -0.5%, 2.03%

TABLE 6.5: Results for Gov2, * and + denotes statistical significance over BM25 and LM, respectively. The best result for each metric is in bold. The percentages below each value is the % improvement over BM25 and LM, respectively

except for P@10. The Gaussian kernel function performs the best overall. Not all of the kernel functions significantly improve BM25 in P@5. However, for the kernel functions that do improve BM25 in P@5, the improvements are statistically significant.

We treat the Uniform kernel function as an outlier and ignore it. The MAP and P@10 performance of our model is consistently significantly better than BM25. The MAP, P@10, and P@20 performance of our model is fairly consistent. Our model also consistently statistically significantly improves LM in MAP and P@5. There are large differences between the highest % improvement over BM25 in P@5 (1.31%) and the lowest (-0.89%). This can be due to the volatility of the P@5 metric. We analyze this further in our robustness study in Chapter 6.2.

We choose two kernel functions that perform well across all collections, Gaussian and Circle, to represent our model. We illustrate the % performance difference between those kernel functions and the baseline weighting models in Figure 6.5.

The figure shows that the performance of our model is better than the baseline weighting models in MAP, P@5, and P@10. The performance of our model is worse than BM25

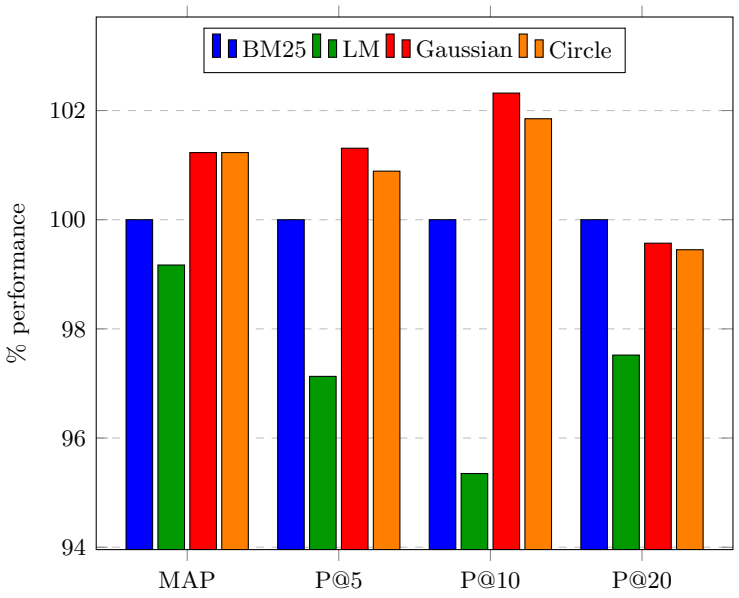

FIGURE 6.5: Gov2, % performance improvement

for documents 11 to 20. This is in contrast with the other collections that we use. The performance of the Gaussian and the Circle kernel functions are almost equal overall.

6.1.6 Overall Effectiveness

The performance of our model is statistically significantly better than BM25 in MAP in every collection and in $P@5$ and $P@10$ in most of the collections. The $P@20$ performance of our model is better than BM25 in most of the collections, but the improvements are only statistically significant in the WT2g and the disk4+5 collections. Our model also performs statistically significantly better than LM in $P@5$ in all collections. Our model performs better than LM in MAP, $P@10$, and $P@20$ in all collections but only some of the improvements are statistically significant. Across all of the collections there is no single kernel function that performs the best. However, the performance of the Uniform kernel function is consistently the worst and is not significantly better than BM25 in MAP, $P@5$, and $P@10$ in any of the collections. The Gaussian and the Circle kernel function performs well across all of the collections.

The reason for the mixed results in the disk4+5 and the WT10g collections seems to be because of poor performance of our model for either documents 1 to 5 or 6 to 10, but not both at the same time. In the Gov2 collection our model also performs poorly for

documents 11 to 20. There seems to be no correlation between the nature of the documents and retrieval performance. For example, both WT2g and WT10g are collections of web documents, yet the the performance of our model, especially in $P@5$ and $P@10$, is vastly different. Overall there are large variances in only the $P@5$ metric. $P@5$ is the most volatile metric, therefore this finding is unsurprising. We analyze this further in our robustness study in Chapter 6.2.

6.2 Parameter Sensitivity

We want to assess the robustness of our model. The more robust a model is, the more likely that it will perform well in an unknown collection without training data. This is important in proving the practicality of a model. In this chapter, we analyze the performance of our model with varying parameters. We have already shown the sensitivity of our model to the collection used through experimentation. The performance of our model is significantly better than BM25 in MAP in all collections and significantly better than LM in MAP in most of the collections. The performance of our models is also significantly better than BM25 and LM in $P@5$ and $P@10$ in most of the collections. Therefore our model is somewhat robust.

We continue to assess the sensitivity of our model to the α, β, and γ parameters in this study. We study the effect of that each parameter has on the performance of our model. We do this by varying one parameter and fixing the other three parameters to the values in the previous experiment. We choose two kernel functions that perform well across all of the collections to represent our model, which are the Gaussian and the Circle kernel functions.

6.2.1 The α Parameter

The α parameter controls the contribution of our model into the score of the documents, where BM25 contributes $(1 - \alpha)\%$ and our model contributes $\alpha\%$. α has a range of [0, 1], where $\alpha = 0$ means that the score of the documents is equivalent to BM25 and $\alpha = 1$ means that BM25 does not contribute to the score of the documents. In this study we vary α in the range of [0.1, 0.9] in steps of 0.1. We fix $k_1 = 1.2$, $k_3 = 8$, $\beta = 3$ and $\gamma = 3$. We set b to the optimal value for each collection in Table 5.2.

6.2.1.1 WT2g

The sensitivity of our model to the α parameter in the WT2g collection is shown in Figure 6.6.

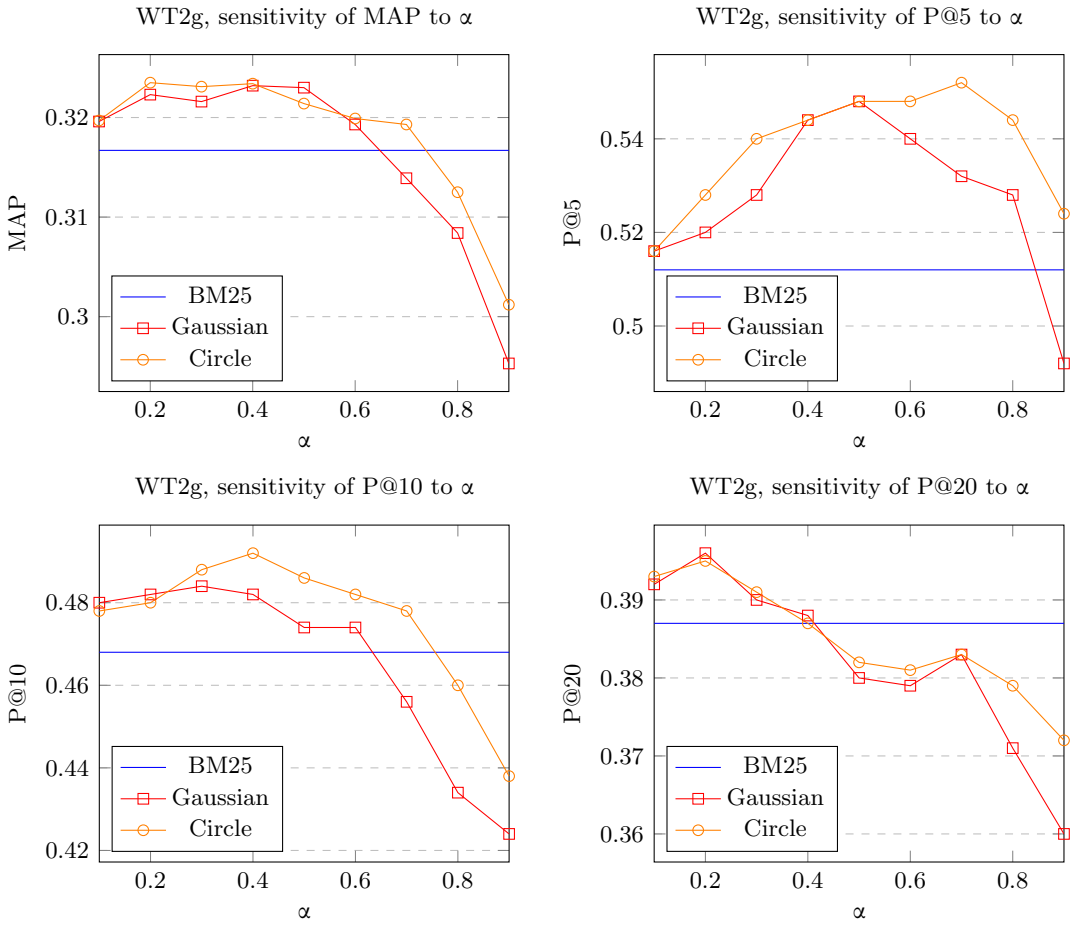

FIGURE 6.6: WT2g, sensitivity to the α parameter

The figure shows that the performance of the two kernel functions react similarly to changes in the value of α. The MAP performance of our model is the best at $\alpha = 0.4$. The MAP performance of our model is fairly consistent for α between [0.2, 0.4], but worsens for $\alpha \geq 0.5$. The $P@5$ performance of our model is the best at $\alpha = 0.5$ for the Gaussian kernel function and at $\alpha = 0.7$ for the Circle kernel function. This is not unexpected because of the volatility of the $P@5$ metric. Our model seems to be very precise in the top 5 documents in the WT2g collection since the Gaussian and the Circle kernel functions out-performs BM25 by 7.0313% at $\alpha = 0.5$ 7.8125% at $\alpha = 0.7$, respectively. If we were to optimize our model for only $P@5$ performance then our model can be very effective. However, $P@5$ performance comes at the cost of MAP, $P@10$,

and P@20 performance, as can be seen in the figure. The P@20 performance of our model is much more sensitive to the α parameter. The P@20 performance of our model is worse than BM25 when $\alpha > 0.4$. Overall our model seems to be somewhat insensitive to the α parameter in the WT2g collection. Although the performance may not be optimal, it is better than BM25 in all metrics when α is between [0.1, 0.4]. Our model is also better in the top 10 documents when α is between [0.1, 0.6].

6.2.1.2 disk4+5

The sensitivity of our model to the α parameter in the disk4+5 collection is shown in Figure 6.7.

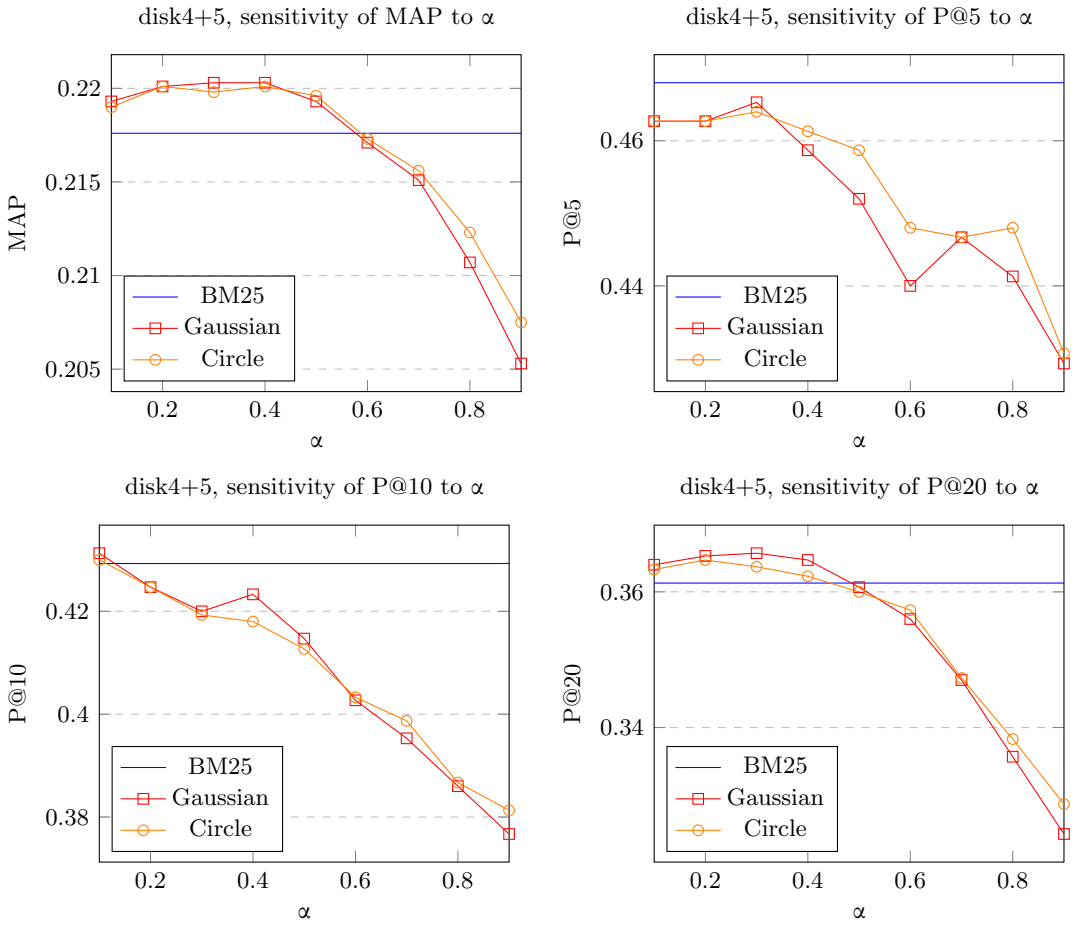

FIGURE 6.7: disk4+5, sensitivity to the α parameter

The figure shows that the performance of the two kernel functions react similarly to changes in the value of α. The MAP performance of our model is the best at $\alpha = 0.4$. The P@10 performance of our model is better than BM25 only if $\alpha = 0.1$. The P@5

performance of our model is the best at $\alpha = 0.3$. However, the $P@5$ performance of our model is worse than BM25 for all values of α. From this study, it seems that our model is generally ineffective in the top 10 documents in the disk4+5 collection. However, it seems to be more effective and robust when the top 20 and more documents are considered.

6.2.1.3 WT10g

The sensitivity of our model to the α parameter in the WT10g collection is shown in Figure 6.8.

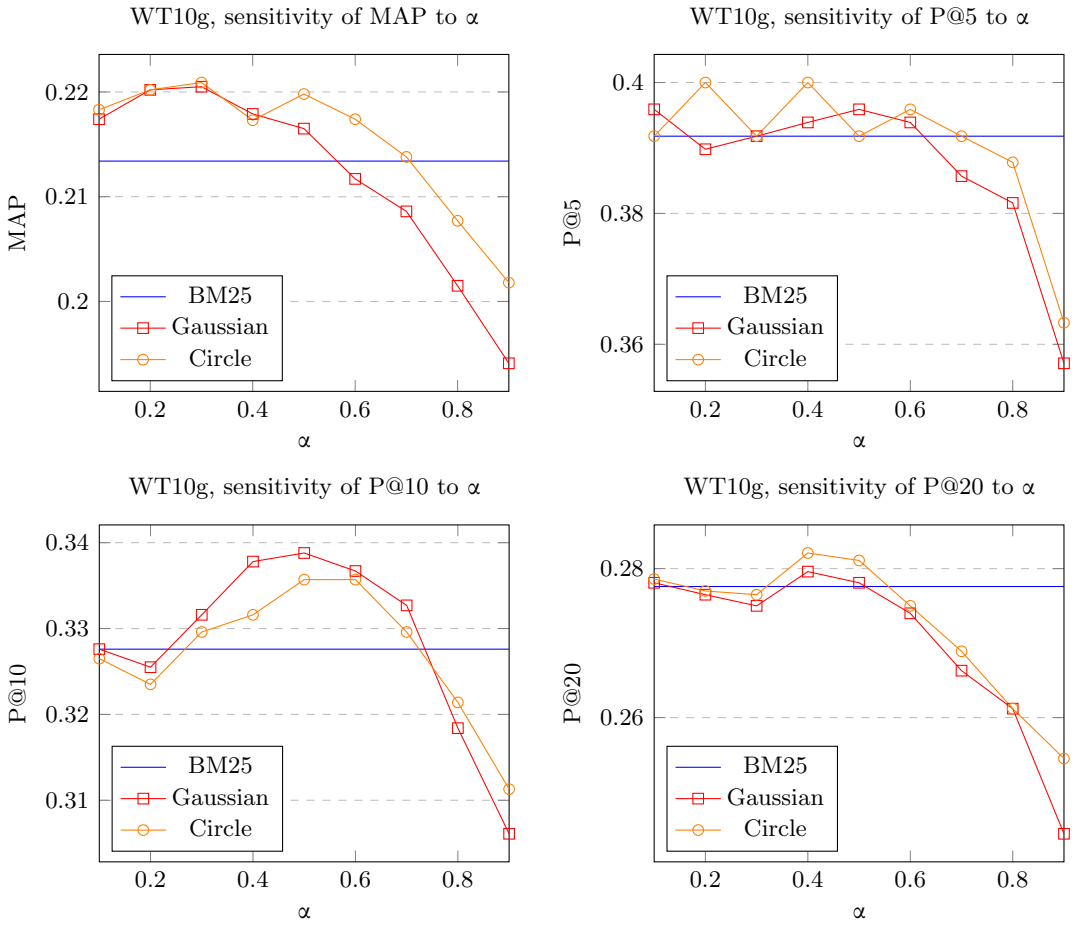

FIGURE 6.8: WT10g, sensitivity to the α parameter

The performance of the two kernel functions are similarly affected by the α parameter. The MAP performance of our model is the best at $\alpha = 0.3$. The $P@5$ performance of our model seems mixed. However, the $P@5$ performance of our model seems to be at least equal to if not better than BM25 when α is between [0.3, 0.6]. The $P@10$

performance of our model is the best at $\alpha = 0.5$. The $P@10$ performance of our model at $\alpha = 0.2$, which is the value which we use for our main experiment, is the worst for $\alpha < 0.8$. Our model performs better than BM25 when α is between [0.4, 0.5]. Overall only the MAP performance of our model is somewhat insensitive to the α parameter in the WT10g collection.

6.2.1.4 Blogs06

The sensitivity of our model to the α parameter in the Blogs06 collection is shown in Figure 6.9.

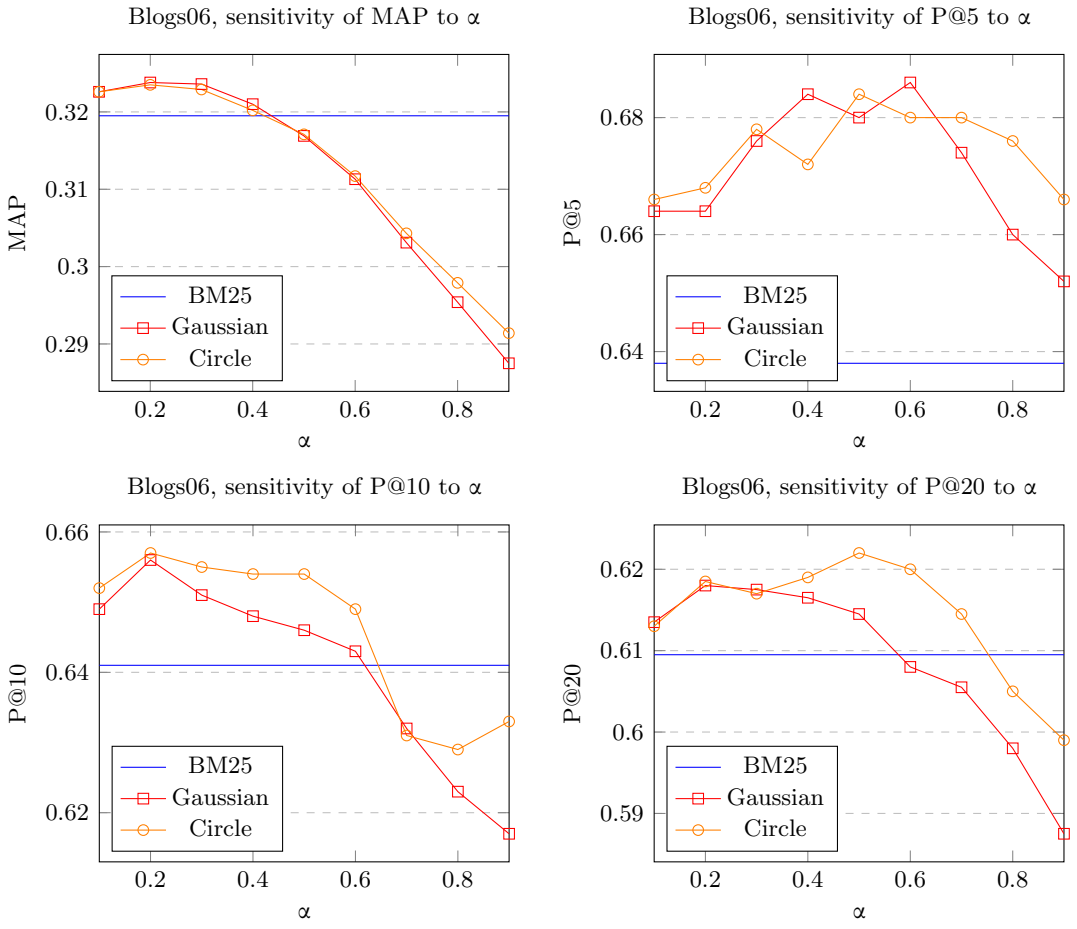

FIGURE 6.9: Blogs06, sensitivity to the α parameter

The figure shows that the performance of the two kernel functions are similar across all of the metrics. Our model performs the best in MAP and $P@5$ when $\alpha = 2$. However, the $P@5$ performance of our model is the best when α is between [0.5, 0.6]. The Gaussian and the Circle kernel functions out-perform BM25 by 7.5235% for $\alpha = 0.6$ and 7.21%

for $\alpha = 0.5$, respectively. This seems to indicate that our model is very well suited for the top 5 documents in the Blogs06 collection. However, this come at the expense of MAP, where our model performs worse than BM25 in MAP when α is between [0.5, 0.6]. Overall our model performs better than BM25 when α is between [0.1, 0.4]. Our model also performs better than BM25 in at least the top 20 documents when α is between [0.1, 0.5]. Therefore our model is somewhat insensitive to the α parameter in the Blogs06 collection.

6.2.1.5 Gov2

The sensitivity of our model to the α parameter in the Gov2 collection is shown in Figure 6.10.

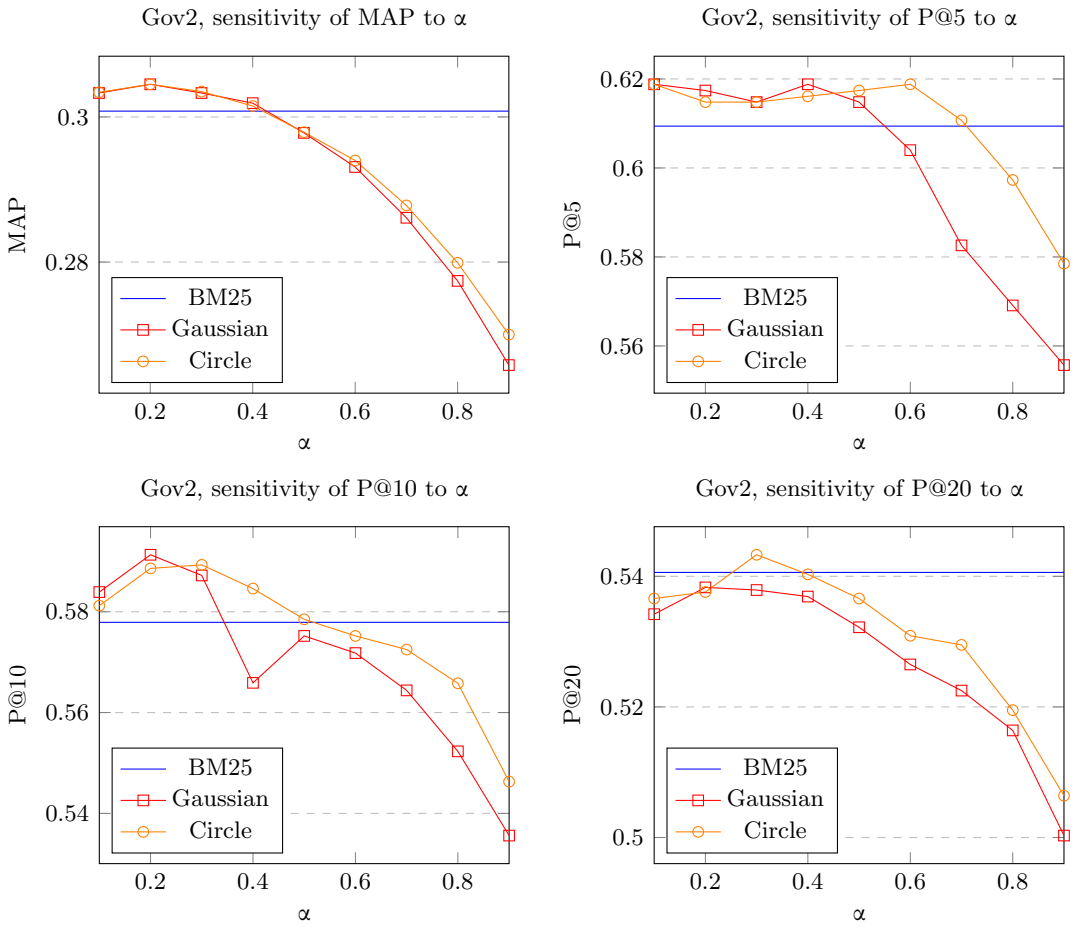

FIGURE 6.10: Gov2, sensitivity to the α parameter

The figure shows that the performance of our model is better than BM25 in MAP for $\alpha \leq 0.4$. The same is true for the $P@5$ and $P@10$ metrics except for the Gaussian kernel

function for $\alpha = 0.4$. The performance of the Gaussian kernel function for $\alpha = 0.4$ seems to be an anomaly since it doesn't follow the performance trend shown in the figure. We will need to investigate whether the same dip in performance occurs when adjusting the other parameters. Overall it seems that our model is well suited for the top 10 documents in the Gov2 collection when α is between [0.1, 0.3]. However, only the Circle kernel function out-performs BM25 in P@20, and only for $\alpha = 0.3$.

6.2.2 The β Parameter

The β parameter is a sentence length dependent parameter that controls the width of the kernel functions. The effect of the β parameter is shown in Chapter 4.1.3. The lower the value of β, the more terms that are given maximum reward. Particularly, if $\beta \leq 2$ and $\gamma = 0$, then every term in the documents are given maximum reward. Therefore β cannot be too low or our model would not be able to effectively discriminate between the important and unimportant terms in the documents. β has no soft or hard upper bound. As β is increased the number of terms that are given maximum reward in a sentence is decreased. In this study we vary β in the range of [2, 9] in steps of 1. We fix $k_1 = 1.2$, $k_3 = 8$, $\alpha = 0.2$ and $\gamma = 3$. We set b to the optimal value for each collection in Table 5.2.

6.2.2.1 WT2g

The sensitivity of our model to the β parameter in the WT2g collection is shown in Figure 6.11.

The figure shows that the MAP, P@5, and P@10 performance of our model is the best at $\beta = 3$. The MAP performance of our model seems to plateau for $\beta \geq 7$. The P@20 performance of our model also plateaus for $\beta \geq 5$. However, performance plateaus are expected with the β parameter since the effect of the β parameter follows the law of diminishing returns. The trend, with the exception of P@20, is that as β is increased past 3, the performance of our model decreases until it eventually plateaus. The P@20 performance of our model is the best when $\beta \geq 5$. This can be an anomaly with the WT2g collection, we will investigate to see if similar performance curves are present in

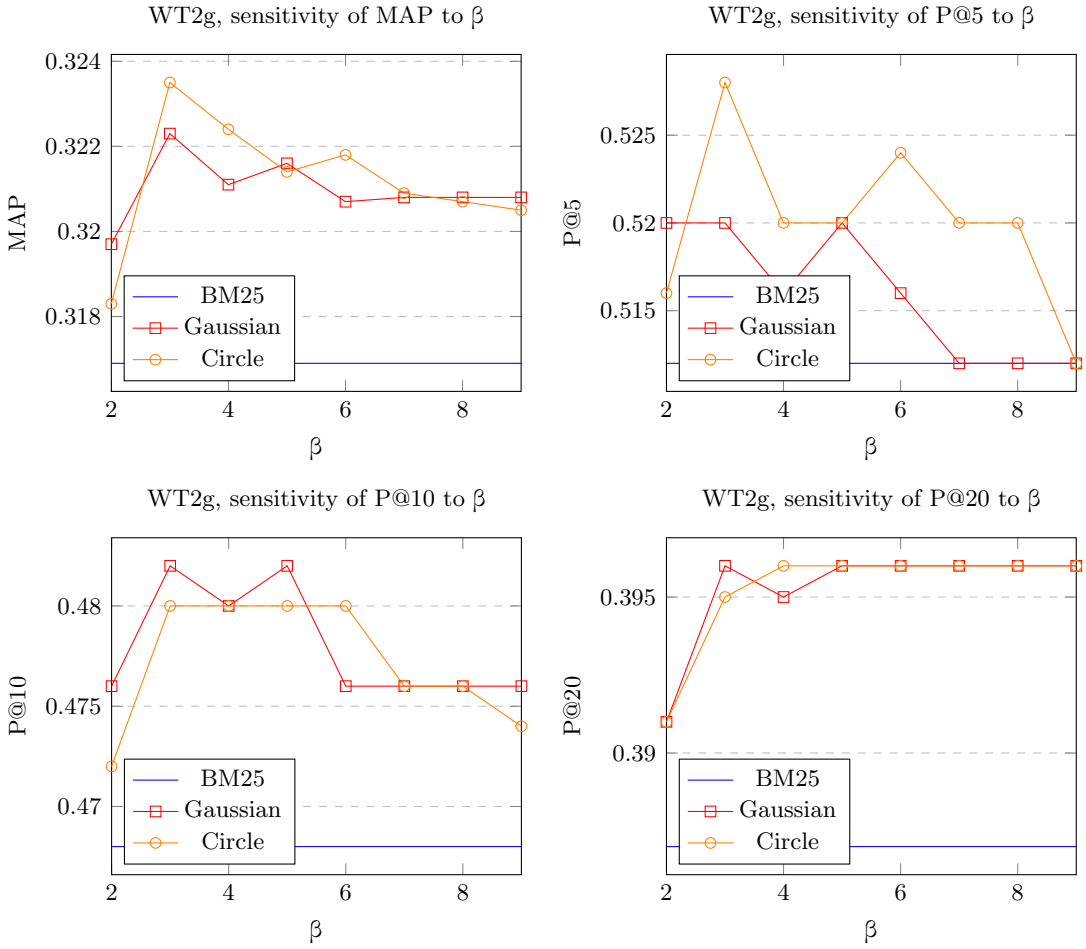

FIGURE 6.11: WT2g, sensitivity to the β parameter

the other collections. Overall it seems that for $\beta \geq 3$, our model is very insensitive to the β parameter in the WT2g collection.

6.2.2.2 disk4+5

The sensitivity of our model to the β parameter in the disk4+5 collection is shown in Figure 6.12.

The figure shows that for both MAP and P@20, the performance of our model is the generally the best at $\beta = 2$. This does not mean that no terms are rewards, since $\gamma = 3$. Rather this means that if the number of terms being rewarded is affected by the length of sentences, then performance starts to decrease. However, this is not true in P@5, where the performance of our model is the best when β is between [0.7, 0.8]. The P@5 and P@10 performance of our model is consistently worse than BM25. It

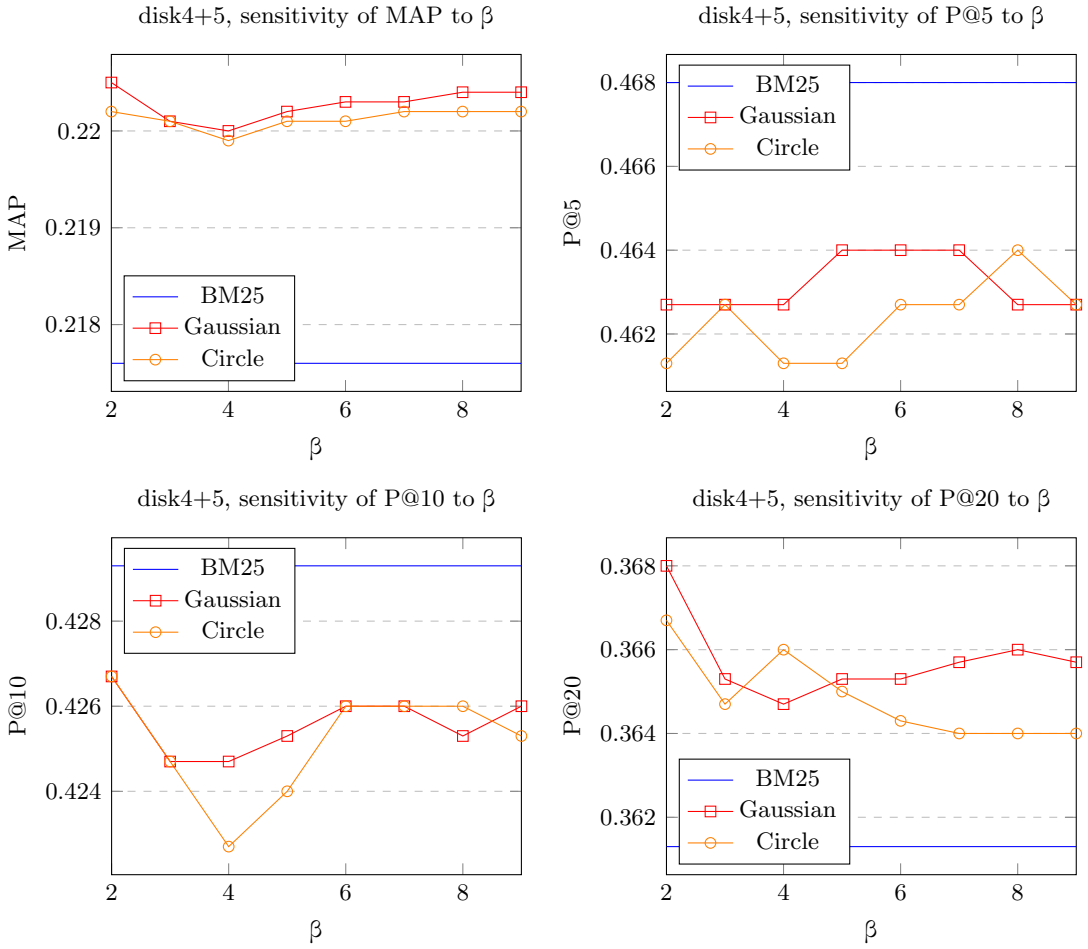

FIGURE 6.12: disk4+5, sensitivity to the β parameter

is fairly conclusive thus far that our model is not effective for the top 10 documents in the disk4+5 collection. However, the MAP and P@20 performance of our model is consistently better than BM25 despite the ineffectiveness of our model for the top 10 documents. Once again, it seems that our model is very insensitive to the β parameter in the disk4+5 collection for $\beta \geq 3$.

6.2.2.3 WT10g

The sensitivity of our model to the β parameter in the WT10g collection is shown in Figure 6.13.

The MAP performance of our model is the best for $\beta = 3$ for the Circle kernel function and for $\beta = 4$ for the Gaussian kernel function. The MAP performance of our model seems to plateau at $\beta \geq 5$. In P@5, our model performs the best for $\beta = 4$. The

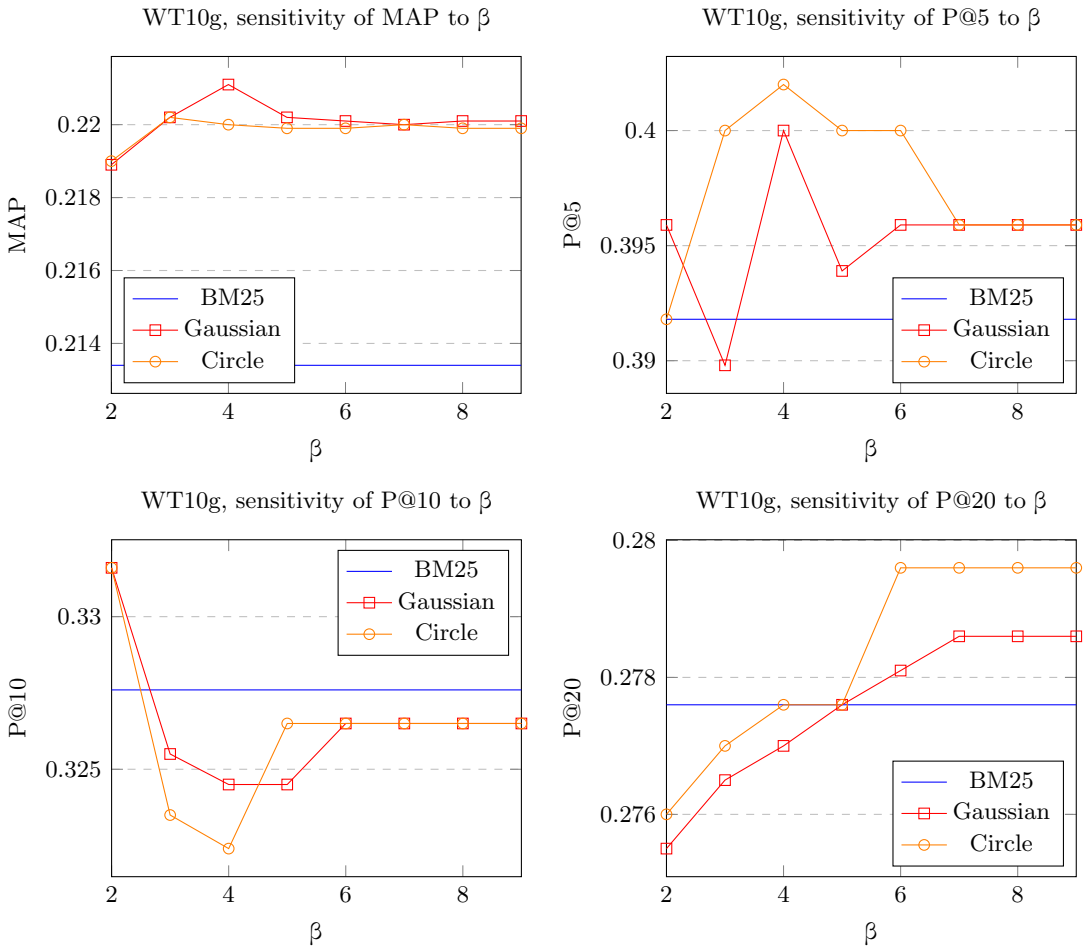

FIGURE 6.13: WT10g, sensitivity to the β parameter

P@5 performance plateaus at $\beta \geq 7$. In P@10, the performance of our model is better than BM25 only for $\beta = 2$. This means that for the top 10 documents in the WT10g collection, our model is the most effective when the number of terms being rewarded does not scale with the length of sentences that the terms are in. In P@20, our model performs the best for $\beta \geq 7$. The P@10 performance of our model is in contrast with the P@5 and P@20 performance of our model, where the P@5 and P@20 performance of our model is the worst for $\beta = 2$. From the figure it seems that our model is ineffective for only documents 6-10 in the WT10g collection.

6.2.2.4 Blogs06

The sensitivity of our model to the β parameter in the Blogs06 collection is shown in Figure 6.14.

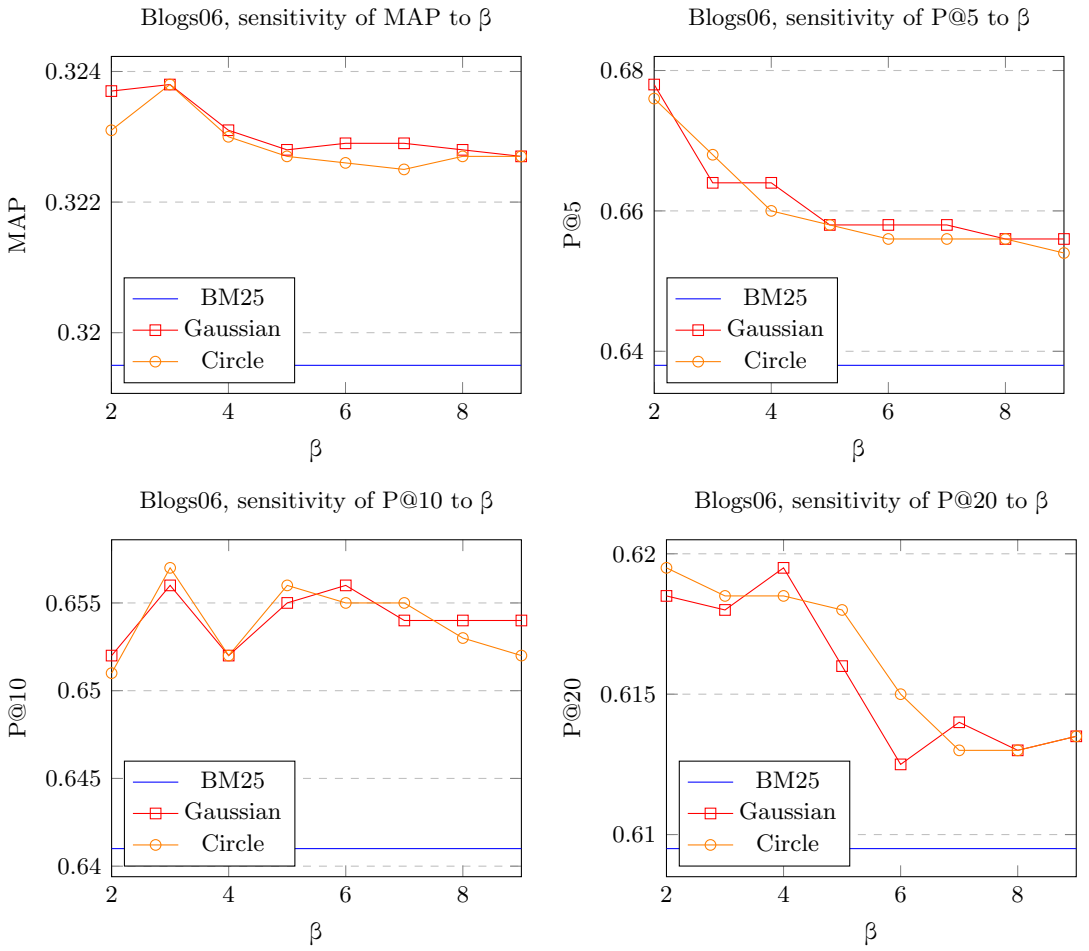

FIGURE 6.14: Blogs06, sensitivity to the β parameter

The trend in MAP, $P@5$, and $P@20$ is similar. The performance of our model is the best or close to the best at $\beta = 2$, but plateaus as β is increased. However our model outperforms BM25 for all values of β. The MAP and $P@10$ performance of our model is the best for $\beta = 3$. The $P@5$ and $P@20$ of the Circle kernel function is the best for $\beta = 2$. The $P@5$ and $P@20$ performance of the Gaussian kernel function is similar to that of the Circle kernel function. Overall there is a clear trend where as β increases, the performance of our model decreases. Our model seems to be robust in the Blogs06 collection for $\beta \leq 4$.

6.2.2.5 Gov2

The sensitivity of our model to the β parameter in the Gov2 collection is shown in Figure 6.15.

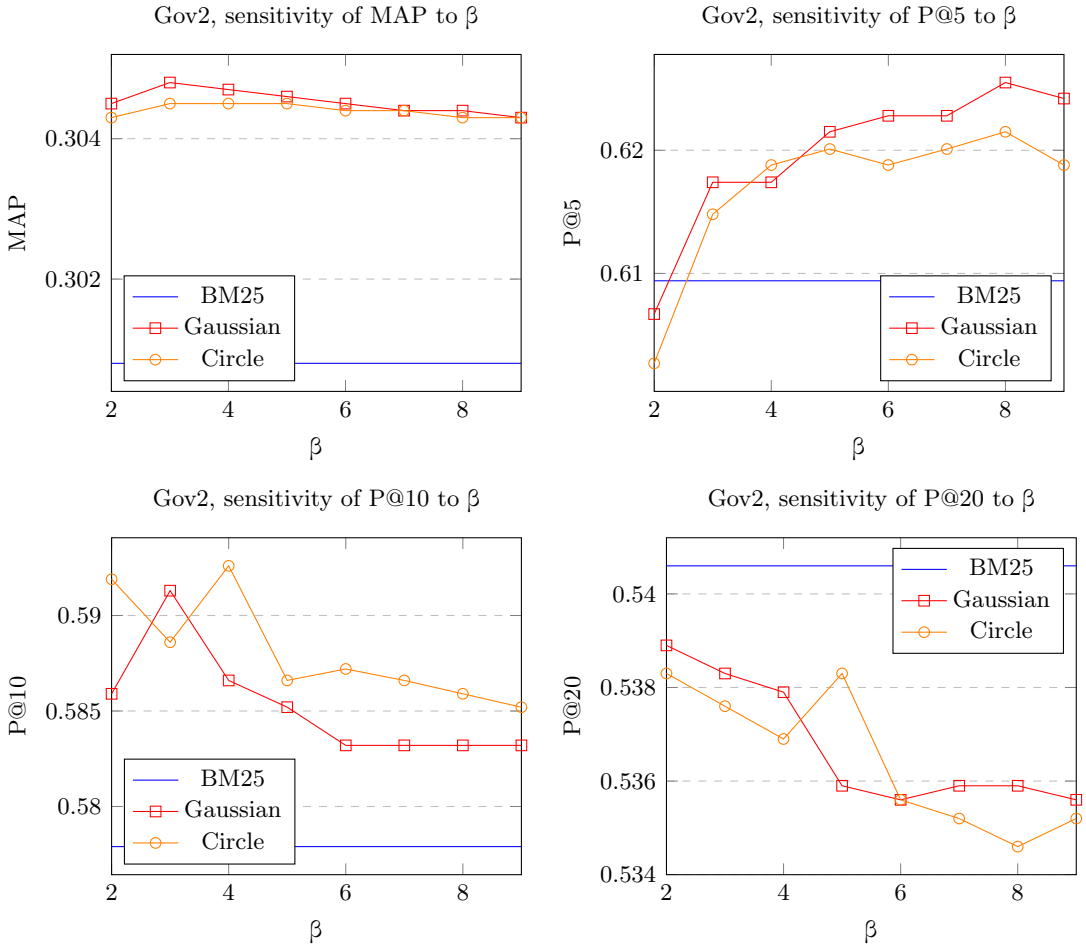

FIGURE 6.15: Gov2, sensitivity to the β parameter

The MAP performance of our model is the best for $\beta = 3$. However, the performance difference between the best MAP performance (0.3048) and the worst MAP performance of our model (0.3043) is only 0.0005. The P@5 performance of our model increases as β is increased and is better than BM25 for $\beta \geq 3$. The P@10 performance of our model is the best for $\beta = 2$ and $\beta = 3$ for the Gaussian and the Circle kernel functions, respectively. The P@10 performance of our model plateaus for $\beta \geq 5$. The P@20 performance of our model is worse than BM25 for all values of β. This seems to indicate that our model is fairly robust for the top 10 documents in the Gov2 collection and overall, given the MAP performance. However our model seems ineffective in at least the top 20 documents, especially if the reward given to terms is scaled with the length of sentences.

6.2.3 The γ Parameter

The γ parameter is a sentence length independent parameter that controls the width of the kernel functions. The effect of the γ parameter is shown in Chapter 4.1.3. The lower the value of γ, the more terms that are given maximum reward. γ has a soft upper bound of $UB = \frac{SL-1}{2}$. If $\gamma = UB$ then the reward given to every term in the sentence is calculated by the kernel function. If $\gamma > UB$ then the previous still applies, but the kernel function is wider, therefore every term in the sentence is given less reward. In this study we vary γ in the range of [0, 9] in steps of 1. We fix $k_1 = 1.2$, $k_3 = 8$, $\alpha = 0.2$ and $\beta = 3$. We set b to the optimal value for each collection in Table 5.2.

6.2.3.1 WT2g

The sensitivity of our model to the γ parameter in the WT2g collection is shown in Figure 6.16.

The figure shows that the two kernel functions react similarly to changes in γ. The MAP and P@5 performance of the Gaussian and the Circle kernel function is the best at $\gamma = 2$ and $\gamma = 3$, respectively. The P@20 performance of our model increases as γ is increased. Since we did not see a similar trend for $\beta \geq 3$ in Figure 6.11, this means that giving less reward to the terms in shorter sentences improves P@20 performance in the WT2g collection. With the exception of the P@20 performance, the general trend is that as γ increases, performance decreases. The performance of our model is the worst or close to the worst for $\gamma = 0$ in all metrics and is worse than BM25 in some cases. This shows that in the WT2g collection, our model is effective only if the terms in shorter sentences are given less reward.

Overall the Gaussian kernel function is very insensitive to the γ parameter if $\gamma \geq 2$. The P@5 performance of our model can be due to the volatility of the metric.

6.2.3.2 disk4+5

The sensitivity of our model to the γ parameter in the disk4+5 collection is shown in Figure 6.17.

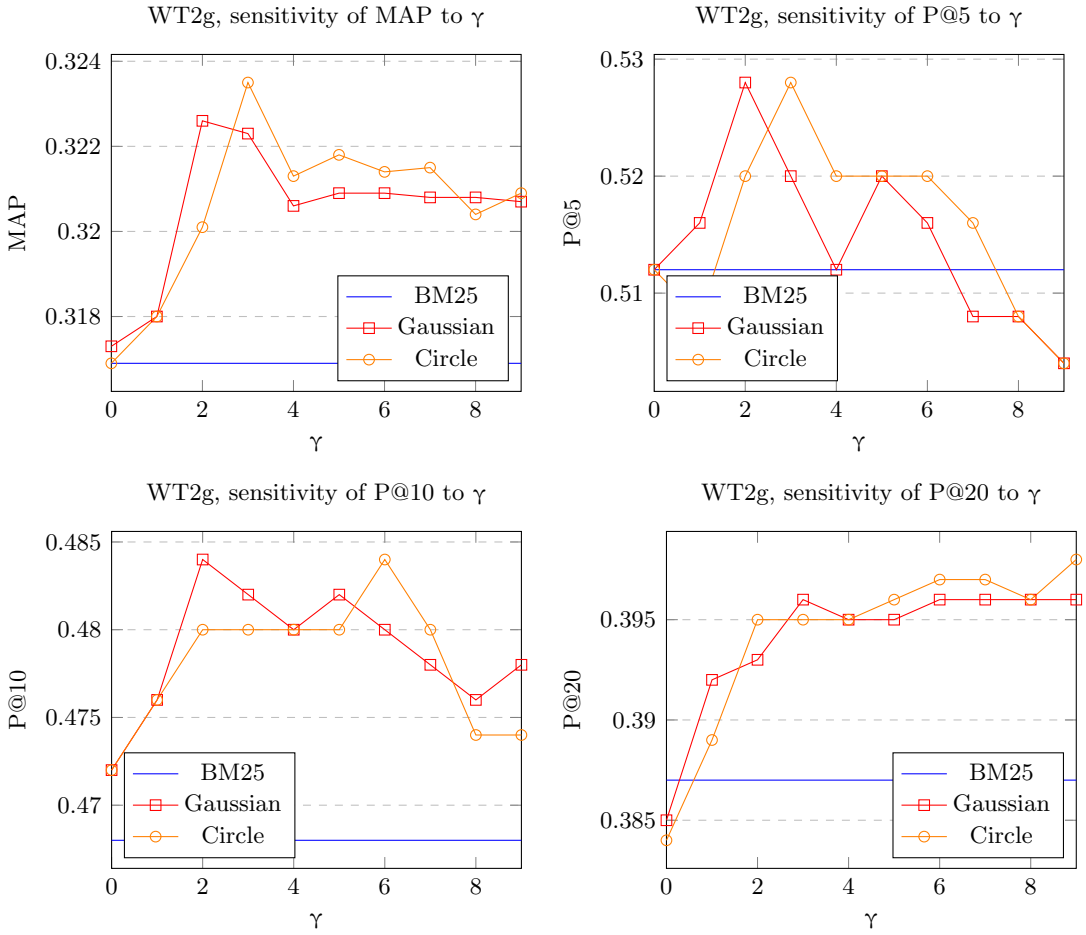

FIGURE 6.16: WT2g, sensitivity to the γ parameter

The MAP and $P@20$ performance of our model peaks at $\gamma = 2$. The MAP performance of our model plateaus at $\gamma \geq 3$. The $P@5$ and $P@10$ performance of our model is the best when $\gamma = 0$. This means that the performance of our model is better when the reward given to terms are more proportional to the length of sentences that the terms are in. It may be worthwhile to investigate negative values for γ in a future study, where we give more reward to terms in shorter sentences. With the exception of documents 6-10, our model seems very insensitive to the γ parameter. Our model performs consistently worse than BM25 in $P@5$, but the performance is comparable for all values of γ.

6.2.3.3 WT10g

The sensitivity of our model to the γ parameter in the WT10g collection is shown in Figure 6.18.

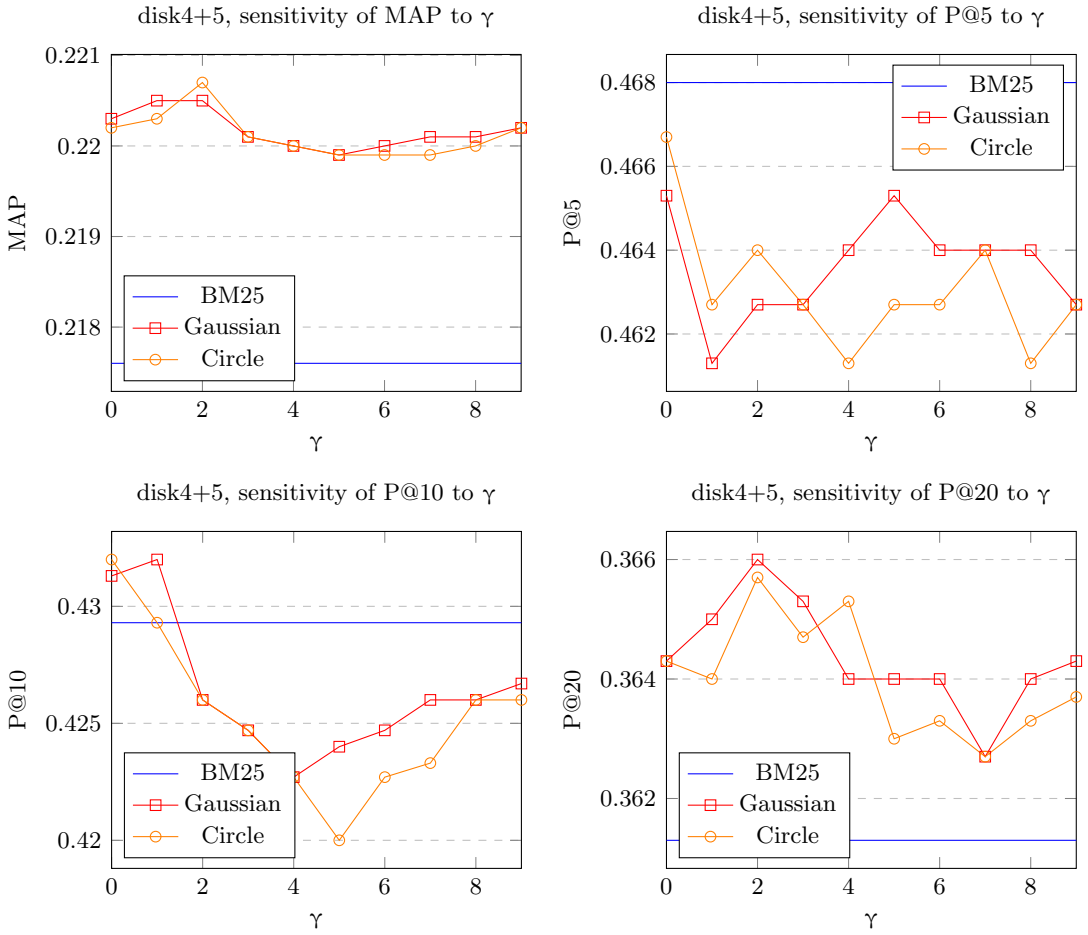

FIGURE 6.17: disk4+5, sensitivity to the γ parameter

The MAP performance of our model is the best at $\gamma = 2$ and $\gamma = 3$ for the Gaussian and the Circle kernel functions, respectively. The P@5 performance of our model is very sporadic, with a sharp dip for the Circle kernel function at $\gamma = 6$. However, P@5 is the most volatile measure used in this experiment. The P@10 performance of our model is the best at $\gamma = 1$ and drops sharply as γ is increased further. The P@20 performance of our model increases as γ is increased and only outperforms BM25 for $\gamma \geq 5$ and $\gamma > 6$ for the Gaussian and the Circle kernel functions, respectively. Overall only the MAP parameter is insensitive to the γ parameter for $\gamma \geq 1$. Our model only outperforms BM25 in P@10 and P@20 for certain values of γ.

6.2.3.4 Blogs06

The sensitivity of our model to the γ parameter in the Blogs06 collection is shown in Figure 6.19.

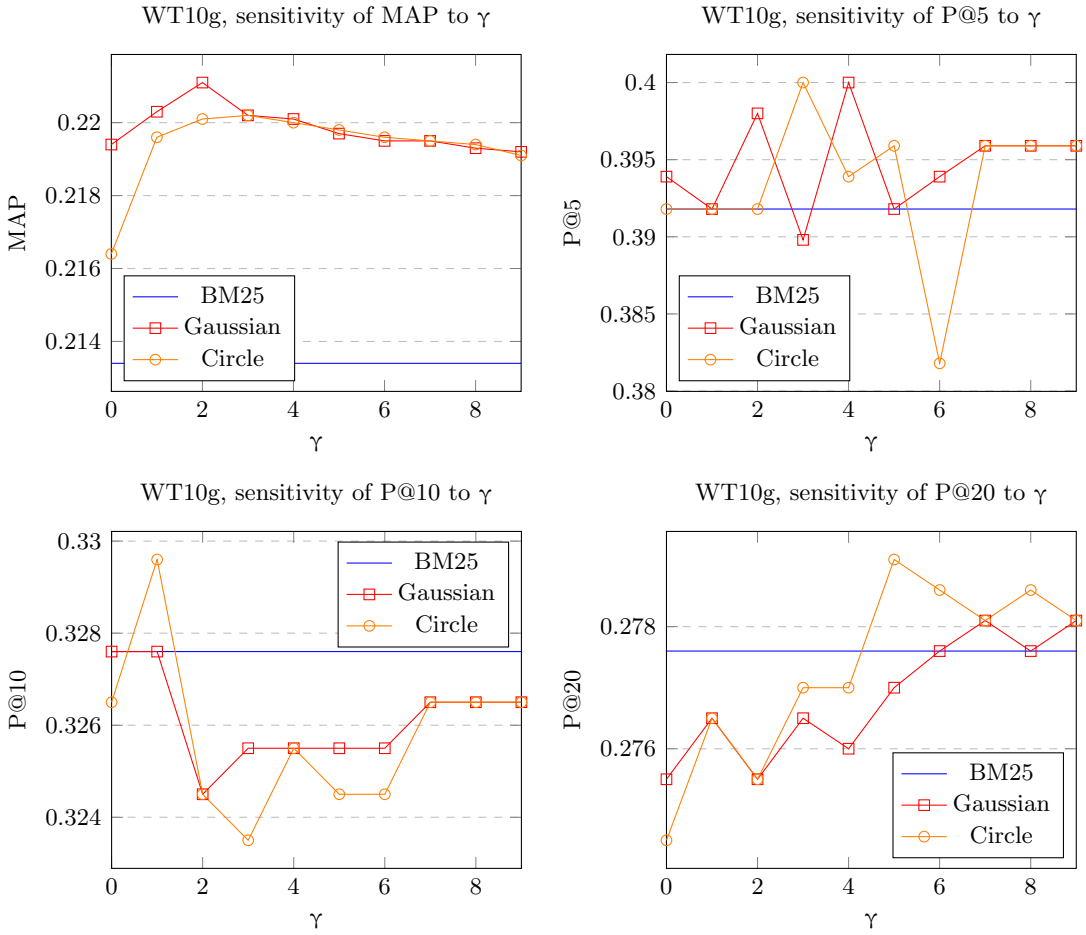

FIGURE 6.18: WT10g, sensitivity to the γ parameter

The MAP performance of our model is the best at $\gamma = 3$ and plateaus for $\gamma \geq 4$. The P@5 performance of our model peaks at $\gamma = 2$ and $\gamma = 3$ for the Gaussian and the Circle kernel functions. The P@5 performance of our model drops as γ is increased. A similar trend can be seen in the P@20 performance of our model. However, our model outperforms BM25 for all values of γ. Overall our model is insensitive to the γ parameter in the Blogs06 collections. The performance of our model seems to be the best when γ is within [1, 3]. However, there is a large difference between the best performance and the worst performance of our model in both P@5 and P@20.

6.2.3.5 Gov2

The sensitivity of our model to the γ parameter in the Gov2 collection is shown in Figure 6.20.

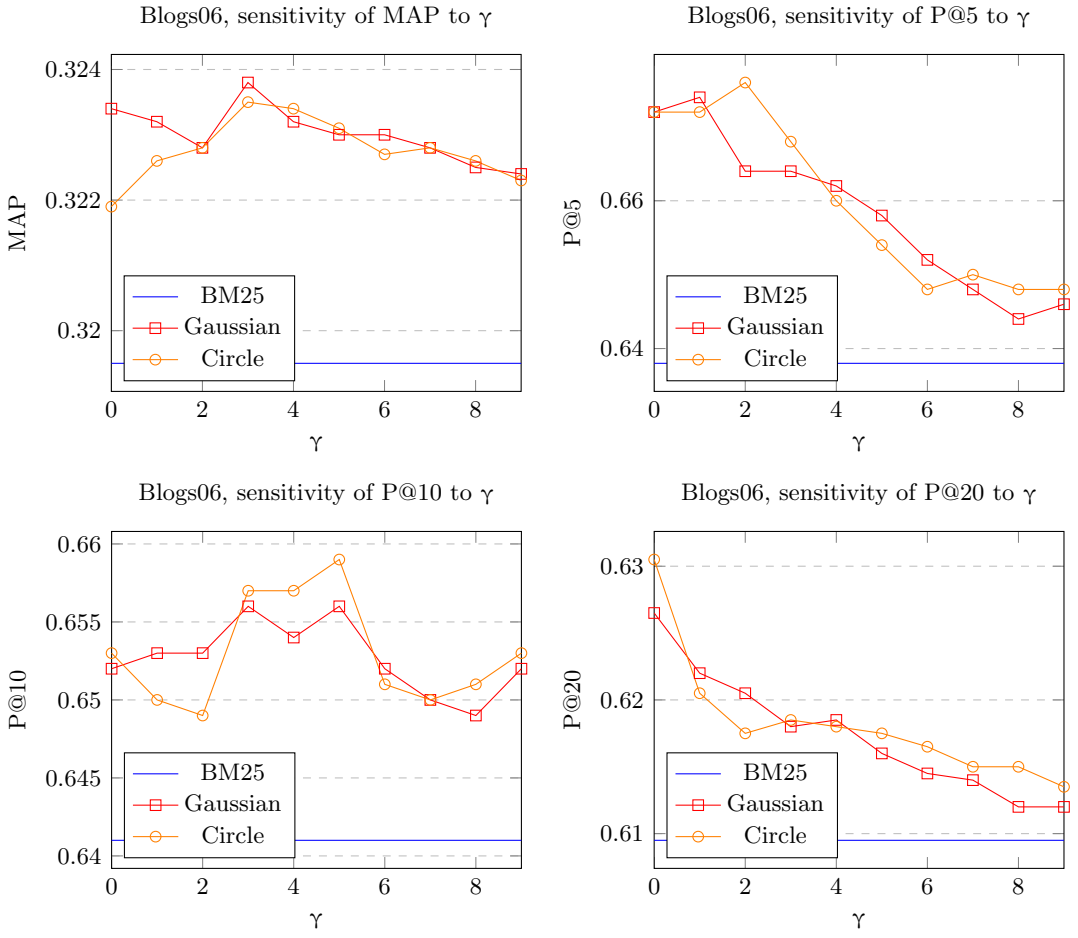

FIGURE 6.19: Blogs06, sensitivity to the γ parameter

The MAP performance of our model is fairly consistent, where our model performs the best for $\gamma = 4$. The $P@5$ performance of our model increases as γ is increased. In $P@10$, there is a distinct peak in performance at $\gamma = 3$ and $\gamma = 4$ for the Gaussian and the Circle kernel functions. Our model performs very sporadically in $P@20$ as γ is changed. Overall our model performs the best when γ is within [3, 4]. Our model is very insensitive to the γ parameter in MAP, but the performance of our model in the other measures seem to rely on certain values of γ.

6.2.4 Summary

The sensitivity of our model to the α parameter varies from collection to collection and from metric to metric. In MAP, the performance of our model is the best when α is between [0.2, 0.3]. The performance of our models for the other metrics is generally the best when α is between [0.4, 0.5]. Performance tends to be worse if α is increased past

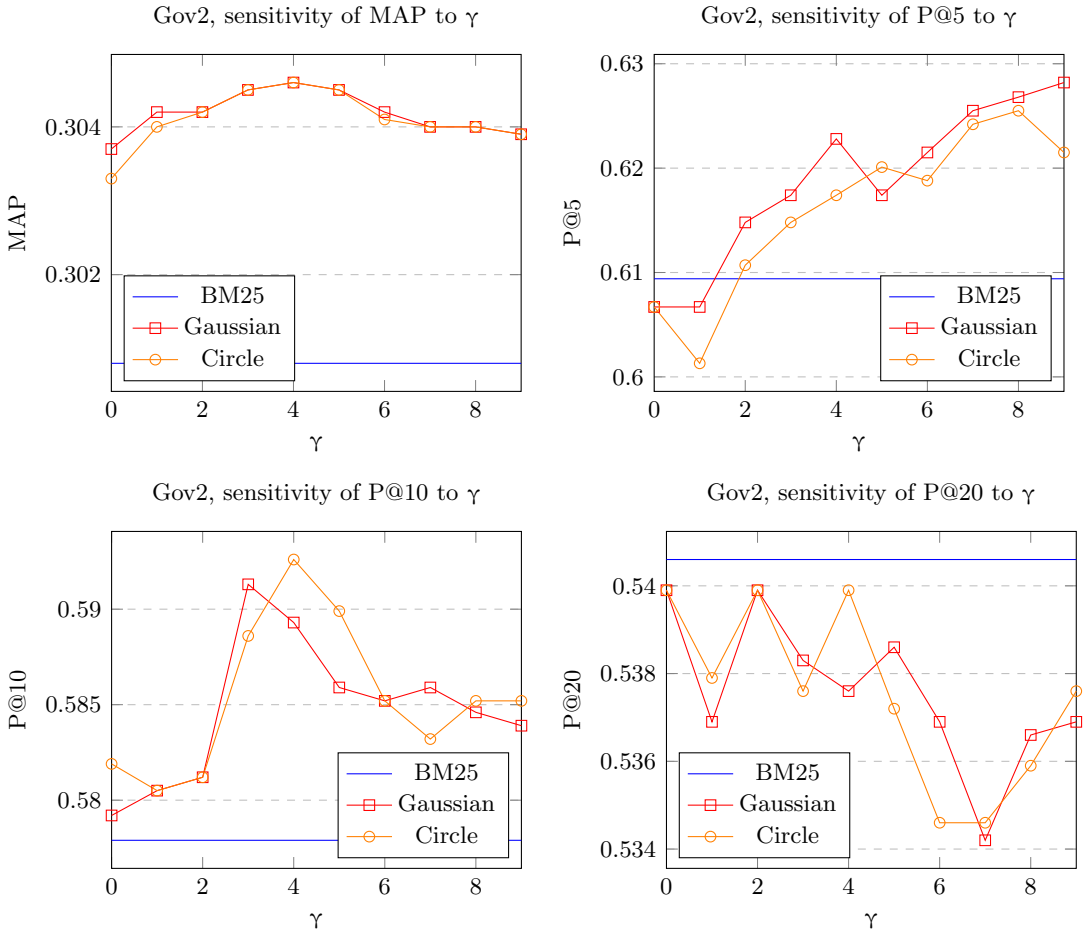

FIGURE 6.20: Gov2, sensitivity to the γ parameter

its optimal value. In some cases there is a trade-off between MAP and the precision of the top documents. Our model is therefore somewhat sensitive to the α parameter. The range that α can vary within without affecting performance significantly is 0.1, which is small.

The effect of the β parameter decreases as it is increased. However, its effect is not linear like the α parameter. Therefore our model should be and is more insensitive to β than it is to α. The MAP performance of our model is the best when $\beta \geq 3$. This means that the maximum reward threshold has almost no effect on MAP past a certain threshold of β. Particularly, with $\beta = 9$ and $\gamma = 3$ very few of the terms are given maximum reward. In some collections the effect of β is also almost negligible on certain kernel functions, such as the Circle kernel function in the Blogs06 collection. In general however, β between [5, 6] is needed for our model to perform the best in P@5, P@10, and P@20.

The effect of the γ parameter is only linear up to the soft upper bound, which varies depending on the length of the sentence. It's worthy to note that certain metrics in some of the collections showed an upward trend as γ is increased and a neutral trend as β is increased. This means that as the performance for some metrics improved as the terms in shorter sentences are given less reward. However, this effect is not global, and can be document-specific. In general, the performance of our model is the best when γ is between [2, 4]. However, for certain cases, such as the P@10 performance in the disk4+5 collection which prefers $\gamma = 0$, this is not true. It may be worthwhile to investigate whether the upward trend continues past 0 into negative values of γ in a future study. In general, our model is somewhat sensitive to the γ parameter, though a range of [2, 4] should suffice in most cases.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis we use the SVO pattern of English sentences [27] to estimate the probability that a term is a noun and important in the document. This is in order to replenish the semantic relationships between the terms that is lost when the bag of words approach is used. The SVO pattern means that the subject and the object are more often placed near the beginning and the end of sentences, respectively [28]. The subjects and the objects are comprised of nouns and noun-phrases, which are more effective in improving IR than the other lexical classes [19, 26, 30–34]. In order to use the term location information, we extend BM25 and reward the terms based on its location in sentences, where a term is that is more likely to be a noun and important in the document is given more reward. In doing this we hope to elevate the documents that the term is more important in, in order to improve IR performance. We propose TEL, a kernel function based model to estimate that probability based on the proximity of the term to sentence-final punctuation. The reward given to terms also undergoes normalization by the length of the sentence, based on the design of our model, and the length of the query, based on our findings in the query term placement preliminary experiment in Chapter 3.3.

Our assumptions are that the nouns and the important terms in the documents are nearer to the beginning and the end of sentences. We confirm that our assumptions are true in our preliminary experiments on the WT2g collection in Chapter 3.2 and 3.3. We

conduct our main experiment on five representative TREC collections that vary both in size and in content. We fix the values of the parameters of our model to $\alpha = 0.2$, $\beta = 3$, and $\gamma = 3$ in order to show that it is robust. Experimental results in Chapter 6 show that the performance of our model is significantly better than BM25 in MAP in every collection. The P@5 and P@10 performance of our model is significantly better than BM25 in almost most of the collections. The P@20 performance of our model is better than BM25 in three of the collections, the improvements are significant in only two of the collections.

We also conduct a robustness study to see the effect that the parameters have on the performance of our model. Our model is somewhat sensitive to all of the parameters. However, the effect of the parameters on MAP and P@20 is small in most of the cases. As more documents are considered, our model becomes more robust. The maximum reward threshold in Equation 4.13 seems to be largely ineffective in improving MAP, but does have an effect on the precision of the top documents. Due to the sensitivity of our model to the parameters, our model may be difficult to optimize. However, experimental results have shows that the performance of our model is significantly better than BM25 in most of the cases even with fixed parameters.

7.2 Future Work

The α, β, and the γ parameters were fixed in our experiments. The results achieved by our model could be even better if those parameters were optimized for each collection. It may also be worthwhile to analyze the optimal value of the α, β, and the γ parameters and the characteristics of the collections to see if a relationship exists. If there is, being able to automatically set the optimal parameter values instead of fixed parameter values would increase the robustness of our model and the feasibility of deploying our model onto a live environment. Similarly, when optimizing the b, k_1, and k_3 parameters of BM25 and the μ parameter of DirichletLM, more fine-grained optimization techniques, such as looking in the neighborhood of the optimal values used, could be applied to improve the performance of both the baselines and our model.

Another possible future extension is to investigate non-symmetric kernel functions and kernel functions with negative values since we find in our query term placement preliminary experiment in Chapter 3.3 that the placement of the terms near the beginning of sentences is different than near the end of sentences. Non-symmetric kernel functions should also alleviate the phrase normalization problem [64]. The $MinDist$ and the $MaxDist$ measures [23] can also improve the performance of our model, since those measures were found to be more effective than the average of the distances between query terms in improving IR. Incorporating the proximity of the terms to periods in order to find hypernymy and meronymy relationships in sentences is another possible extension to our model.

In our experiments we only compared the performance of our model against BM25 and DirichletLM. The experiments showed that there were significant improvements over the baseline models in all datasets. However, we would still need to compare the performance of our model to current state of the art retrieval models in order to see if the performance improvements are good enough. Lastly, this model could also be used to extend a language IR model such as DirichletLM.

Bibliography

[1] Amit Singhal. Modern information retrieval: A brief overview. *IEEE Data Eng. Bull.*, 24(4):35–43, 2001.

[2] Vannevar Bush. As we may think. 1945.

[3] Hans Peter Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4):309–317, 1957.

[4] Gerard Salton, Chung-Shu Yang, and CLEMENT T Yu. A theory of term importance in automatic text analysis. *Journal of the American society for Information Science*, 26(1):33–44, 1975.

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

[6] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[7] Stephen E Robertson and K Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

[8] Jay M Ponte and W Bruce Croft. A language modeling approach to information retrieval. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 275–281. ACM, 1998.

[9] John Lafferty and Chengxiang Zhai. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 111–119. ACM, 2001.

[10] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to ad hoc information retrieval. In *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 334–342. ACM, 2001.

[11] Stephen E Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 232–241. Springer-Verlag New York, Inc., 1994.

[12] Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *NIST SPECIAL PUBLICATION SP*, pages 109–109, 1995.

[13] Stephen E Robertson, Steve Walker, MM Beaulieu, Mike Gatford, and Alison Payne. Okapi at trec-4. In *Proceedings of the fourth text retrieval conference*, pages 73–97. NIST Special Publication, 1996.

[14] Stephen Robertson and Hugo Zaragoza. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[15] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.

[16] Stephen Paul Harter. *A probabilistic approach to automatic keyword indexing*. PhD thesis, University of Chicago, 1974.

[17] Stephen E Robertson, CJ Van Rijsbergen, and Martin F Porter. Probabilistic models of indexing and searching. In *Proceedings of the 3rd annual ACM conference on Research and development in information retrieval*, pages 35–56. Butterworth & Co., 1980.

[18] Martin F Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.

[19] Mandar Mitra, Chris Buckley, Amit Singhal, Claire Cardie, et al. An analysis of statistical and syntactic phrases. In *RIAO*, volume 97, pages 200–214, 1997.

[20] Yves Rasolofo and Jacques Savoy. Term proximity scoring for keyword-based retrieval systems. In *Advances in Information Retrieval*, pages 207–218. Springer, 2003.

[21] Stefan Büttcher, Charles LA Clarke, and Brad Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 621–622. ACM, 2006.

[22] Desislava Petkova and W Bruce Croft. Proximity-based document representation for named entity retrieval. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management*, pages 731–740. ACM, 2007.

[23] Tao Tao and ChengXiang Zhai. An exploration of proximity measures in information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 295–302. ACM, 2007.

[24] Jiashu Zhao, Jimmy Xiangji Huang, and Ben He. CRTER: using cross terms to enhance probabilistic information retrieval. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 155–164. ACM, 2011.

[25] Yadong Zhu, Yuanhai Xue, Jiafeng Guo, Yanyan Lan, Xueqi Cheng, and Xiaoming Yu. Exploring and exploiting proximity statistic for information retrieval model. In *Information Retrieval Technology*, pages 1–13. Springer, 2012.

[26] Yufeng Jing and W Bruce Croft. An association thesaurus for information retrieval. In *Proceedings of RIAO*, volume 94, pages 146–160. Citeseer, 1994.

[27] Charles F Meyer et al. *Introducing English Linguistics International Student Edition*. Cambridge University Press, 2010.

[28] Hogue Ann. The essentials of english-a writer's handbook, 2003.

[29] David Hawking and Paul Thistlewaite. Proximity operators-so near and yet so far. In *Proceedings of the 4th Text Retrieval Conference*, pages 131–143, 1995.

[30] David A Evans and Chengxiang Zhai. Noun-phrase analysis in unrestricted text for information retrieval. In *Proceedings of the 34th annual meeting on Association for Computational Linguistics*, pages 17–24. Association for Computational Linguistics, 1996.

[31] Shuang Liu, Fang Liu, Clement Yu, and Weiyi Meng. An effective approach to document retrieval via utilizing wordnet and recognizing phrases. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 266–272. ACM, 2004.

[32] Hai-Tao Zheng, Bo-Yeong Kang, and Hong-Gee Kim. Exploiting noun phrases and semantic relationships for text document clustering. *Information Sciences*, 179(13):2249–2262, 2009.

[33] Ken Barker and Nadia Cornacchia. Using noun phrase heads to extract document keyphrases. In *Advances in Artificial Intelligence*, pages 40–52. Springer, 2000.

[34] Joe Fagan. Automatic phrase indexing for document retrieval. In *Proceedings of the 10th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 91–101. ACM, 1987.

[35] Eric Brill. A simple rule-based part of speech tagger. In *Proceedings of the workshop on Speech and Natural Language*, pages 112–116. Association for Computational Linguistics, 1992.

[36] Dan Yang, Christina Leber, Luis Tari, Aravind Chandramouli, Andrew Crapo, Richard Messmer, and Steven Gustafson. A natural language processing and semantic-based system for contract analysis. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on*, pages 707–712. IEEE, 2013.

[37] K Liu, WW Chapman, G Savova, CG Chute, N Sioutos, and Rebecca S Crowley. Effectiveness of lexico-syntactic pattern matching for ontology enrichment with clinical documents. *Methods of information in medicine*, 50(5):397, 2011.

[38] Sheng-Hao Hung, Chia-Hung Lin, and Jen-Shin Hong. Web mining for event-based commonsense knowledge using lexico-syntactic pattern matching and semantic role labeling. *Expert Systems with Applications*, 37(1):341–347, 2010.

[39] Fidelia Ibekwe-SanJuan, Fernandez Silvia, Sanjuan Eric, and Charton Eric. Annotation of scientific summaries for information retrieval. *arXiv preprint arXiv:1110.5722*, 2011.

[40] Andrew Trotman. Choosing document structure weights. *Information Processing & Management*, 41(2):243–264, 2005.

[41] Zhicheng Dou, Sha Hu, Yulong Luo, Ruihua Song, and Ji-Rong Wen. Finding dimensions for queries. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1311–1320. ACM, 2011.

[42] Qinmin Hu, Jimmy Xiangji Huang, and Xiaohua Hu. Modeling and mining term association for improving biomedical information retrieval performance. *BMC bioinformatics*, 13(9):1, 2012.

[43] Atanaz Babashzadeh, Jimmy Huang, and Mariam Daoud. Exploiting semantics for improving clinical information retrieval. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 801–804. ACM, 2013.

[44] Cheng Hua Li and Jimmy Xiangji Huang. Spam filtering using semantic similarity approach and adaptive bpnn. *Neurocomputing*, 92:88–97, 2012.

[45] Xiaofeng Zhou, Jimmy Xiangji Huang, and Ben He. Enhancing ad-hoc relevance weighting using probability density estimation. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 175–184. ACM, 2011.

[46] Fanghong Jian, Jimmy Xiangji Huang, Jiashu Zhao, Tingting He, and Po Hu. A simple enhancement for ad-hoc information retrieval via topic modelling. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 733–736. ACM, 2016.

[47] Xinhui Tu, Jimmy Xiangji Huang, Jing Luo, and Tingting He. Exploiting semantic coherence features for information retrieval. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 837–840. ACM, 2016.

[48] Zheng Ye and Jimmy Xiangji Huang. A learning to rank approach for quality-aware pseudo-relevance feedback. *Journal of the Association for Information Science and Technology*, 2015.

[49] Jun Miao, Jimmy Xiangji Huang, and Jiashu Zhao. TopPRF: A probabilistic framework for integrating topic space into pseudo relevance feedback. *ACM Transactions on Information Systems (TOIS)*, 34(4):22, 2016.

[50] Jiashu Zhao, Jimmy Xiangji Huang, and Shicheng Wu. Rewarding term location information to enhance probabilistic information retrieval. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 1137–1138. ACM, 2012.

[51] Jangwon Seo and Jiwoon Jeon. High precision retrieval using relevance-flow graph. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 694–695. ACM, 2009.

[52] Andreas Broschart and Ralf Schenkel. Proximity-aware scoring for xml retrieval. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 845–846. ACM, 2008.

[53] Jiashu Zhao and Jimmy Xiangji Huang. An enhanced context-sensitive proximity model for probabilistic information retrieval. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 1131–1134. ACM, 2014.

[54] Jiashu Zhao, Jimmy Xiangji Huang, and Zheng Ye. Modeling term associations for probabilistic information retrieval. *ACM Transactions on Information Systems (TOIS)*, 32(2):7, 2014.

[55] Parvaz Mahdabi, Shima Gerani, Jimmy Xiangji Huang, and Fabio Crestani. Leveraging conceptual lexicon: query disambiguation using proximity information for patent retrieval. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 113–122. ACM, 2013.

[56] Baiyan Liu, Xiangdong An, and Jimmy Xiangji Huang. Using term location information to enhance probabilistic information retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 883–886. ACM, 2015.

[57] Ben He, Jimmy Xiangji Huang, and Xiaofeng Zhou. Modeling term proximity for probabilistic information retrieval models. *Information Sciences*, 181(14):3017–3031, 2011.

[58] Ruihua Song, Michael J Taylor, Ji-Rong Wen, Hsiao-Wuen Hon, and Yong Yu. Viewing term proximity from a different perspective. In *Advances in Information Retrieval*, pages 346–357. Springer, 2008.

[59] Bong-Hyun Cho, Changki Lee, and Gary Geunbae Lee. Exploring term dependences in probabilistic information retrieval model. *Information processing & management*, 39(4):505–519, 2003.

[60] Jun Miao, Jimmy Xiangji Huang, and Zheng Ye. Proximity-based rocchio's model for pseudo relevance. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, pages 535–544. ACM, 2012.

[61] Michel Beigbeder and Annabelle Mercier. An information retrieval model using the fuzzy proximity degree of term occurences. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1018–1022. ACM, 2005.

[62] Yuanhua Lv and ChengXiang Zhai. Positional language models for information retrieval. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 299–306. ACM, 2009.

[63] Huda Mohammed Barakat, Maizatul Akmar Ismail, and Sri Devi Ravana. Utilization of cross-terms to enhance the language model for information retrieval. *Malaysian Journal of Computer Science*, 26(3), 2013.

[64] Seeger Fisher and Brian Roark. Query-focused summarization by supervised sentence ranking and skewed word distributions. In *Proceedings of the Document Understanding Conference, DUC-2006, New York, USA*, 2006.

[65] Hans Peter Luhn. The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165, 1958.

[66] Jiaul H Paik. A novel tf-idf weighting scheme for effective ranking. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, pages 343–352. ACM, 2013.

[67] Charles Dickens. Hard times by charles dickens, 1997. URL http://www.gutenberg.org/files/786/786-0.txt. [Online; accessed 5-March-2014].

[68] The Stanford Natural Language Processing Group. Stanford log-linear part-of-speech tagger 3.3.1, 2015. URL http://nlp.stanford.edu/software/stanford-postagger-2014-01-04.zip. [Online; accessed 20-June-2015].

[69] Iadh Ounis, Maarten de Rijke, Craig Macdonald, Gilad Mishne, and Ian Soboroff. Overview of the trec-2006 blog track.

[70] Ellen M Voorhees and Donna Harman. Overview of the sixth text retrieval conference (trec-6). *Information Processing & Management*, 36(1):3–35, 2000.

[71] Amanda Spink, Dietmar Wolfram, Major BJ Jansen, and Tefko Saracevic. Searching the web: The public and their queries. *Journal of the American society for information science and technology*, 52(3):226–234, 2001.

[72] Bernard J Jansen, Amanda Spink, Judy Bateman, and Tefko Saracevic. Real life information retrieval: A study of user queries on the web. In *ACM SIGIR Forum*, volume 32, pages 5–17. ACM, 1998.

[73] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. Analysis of a very large web search engine query log. In *ACm SIGIR Forum*, volume 33, pages 6–12. ACM, 1999.

[74] Iadh Ounis, Gianni Amati, Vassilis Plachouras, Ben He, Craig Macdonald, and Douglas Johnson. Terrier information retrieval platform. In *Advances in Information Retrieval*, pages 517–519. Springer, 2005.

[75] The University of Pennsylvania. The university of pennsylvania (penn) treebank tag-set, 2009. URL http://www.comp.leeds.ac.uk/amalgam/tagsets/upenn.html. [Online; accessed 15-June-2015].

[76] H2 Database Engine. H2 database engine 1.3.174, 2015. URL http://repo2.maven.org/maven2/com/h2database/h2/1.3.174/h2-1.3.174.jar. [Online; accessed 20-June-2015].

[77] Martin F Porter. Porter stemmer in java, 2007. URL http://tartarus.org/martin/PorterStemmer/java.txt. [Online; accessed 15-June-2015].

[78] Terrier Team. Terrier ir platform 3.5, 2015. URL http://terrier.org/download/agree.shtml?terrier-3.5.tar.gz. [Online; accessed 20-June-2015].

Appendix A

Placement of Nouns in Sentences Preliminary Experiment

A.1 Instructions

In order to run the Place of Nouns in Sentences preliminary experiment in Chapter 3.2, you need to do the following:

1. Copy and paste the Noun Placement Parser code in Section A.2 into a file named Parser.java.
2. Place the .jar file from Stanford Tagger 3.3.1 (stanford-postagger-3.3.1.jar) [68] into the same directory as Parser.java.
3. Compile the Noun Placement Parser by entering "javac -cp .:stanford-postagger-3.3.1.jar Parser.java" into the command line.
4. Place the english-left3words-distsim.tagger file from Stanford Tagger 3.3.1 [68] into the same directory as Parser.java.
5. Place the collection directory into the same directory as Parser.java. Rename the collection directory according to the collection as follows: WT2g to wt2g, disk4+5 to disk4+5, WT10g to wt10g, Blogs06 to blogs06, Gov2 to gov2.
6. Run the Noun Placement Parser by entering "java -cp .:standford-postagger-3.3.1.jar Parser [Data] [PoS]" into the command line. Both the [Data] and [PoS] parameters

are mandatory. [Data] is the name of the collection directory, ex. wt2g. [PoS] is the part-of-speech to find (ex. NN for nouns, see [75] for the full set of part-of-speech tags). For example, to find the nouns in the WT2G collection, enter "java -cp .:stanford-postagger-3.3.1.jar Parser wt2g NN" into the command line.

The Noun Placement Parser only supports the collections outlined in Section 5.1. Other collections are not guaranteed to work.

A.2 Noun Placement Parser

```
1  // javac -cp .:stanford-postagger-3.3.1.jar Parser.java
2  // java -cp .:stanford-postagger-3.3.1.jar Parser data pos
3
4  import java.io.BufferedReader;
5  import java.io.File;
6  import java.io.FileReader;
7  import java.io.IOException;
8  import java.util.Arrays;
9  import java.util.ArrayList;
10 import java.util.HashMap;
11 import java.util.HashSet;
12 import java.util.regex.Matcher;
13 import java.util.regex.Pattern;
14 import edu.stanford.nlp.tagger.maxent.MaxentTagger;
15
16 public class Parser
17 {
18   private static double noun_sum = 0d, noun_sum_l = 0d, noun_sum_r = 0d;
19   private static long sen_sum = 0L, noun_count = 0L, noun_count_l = 0L,
       noun_count_r = 0L, sen_count = 0L;
20
21   private static final int min_sen_length = 7, max_sen_length = 20,
       min_sen_thresh = min_sen_length - 1, max_sen_thresh = max_sen_length
       + 1, min_line_thresh = min_sen_length * 2 - 2, max_line_thresh =
       max_sen_length * 100 + 1;
22
23   public static void main (String[] args)
24   {
```

```
25      MaxentTagger tagger = new
        MaxentTagger("english-left3words-distsim.tagger");
26
27      Matcher matcher;
28      // Parse output from tagger
29      Pattern pattern_tag = Pattern.compile("([^-\\s_]+)_([a-zA-Z\\.]+)");
30
31      // Delimiters for text in documents
32      HashMap<String, String> delims = new HashMap<>();
33      String[] dir;
34      String line, start, end;
35      boolean read = false;
36
37      delims.put("wt2g_start", "</DOCHDR>");
38      delims.put("wt2g_end", "</DOC>");
39      delims.put("disk4+5_start", "<TEXT>");
40      delims.put("disk4+5_end", "</TEXT>");
41      delims.put("wt10g_start", "</DOCHDR>");
42      delims.put("wt10g_end", "</DOC>");
43      delims.put("blogs06_start", "</DOCHDR>");
44      delims.put("blogs06_end", "</DOC>");
45      delims.put("gov2_start", "</DOCHDR>");
46      delims.put("gov2_end", "</DOC>");
47
48      start = delims.get(args[0] + "_start");
49      end = delims.get(args[0] + "_end");
50
51      try
52      {
53        dir = (new File(args[0])).list();
54        Arrays.sort(dir);
55        for (String d1 : dir)
56        {
57
58          dir = (new File(args[0] + "/" + d1)).list();
59          Arrays.sort(dir);
60          for (String d2 : dir)
61          {
62            System.out.println(d1 + "/" + d2);
63
```

```
64          try (BufferedReader r = new BufferedReader(new
    FileReader(args[0] + "/" + d1 + "/" + d2)))
65          {
66            while ((line = r.readLine()) != null)
67            {
68              if (read && !line.contains(end))
69              {
70                // Define end of line as end of sentence and parse
71                line =
    line.replaceAll("<[^>]*>|[^\u0020-\u002F\u003A-\u007F]", "") + " ! ";
72                if (line.length() > min_line_thresh && line.length() <
    max_line_thresh)
73                {
74                  ParseLine(pattern_tag.matcher(tagger.tagString(line)),
    args[1]);
75                }
76              }
77              else if (read)
78              {
79                read = false;
80              }
81              else if (line.contains(start))
82              {
83                read = true;
84              }
85            }
86          }
87
88          System.out.println("\nAvg dist: " + (noun_sum / noun_count) +
    "\nAvg dist left: " + (noun_sum_l / noun_count_l) + "\nAvg dist
    right: " + (Math.abs(noun_sum_r) / noun_count_r) + "\nNouns: " +
    noun_count + "\nNouns left: " + noun_count_l + "\nNouns right: " +
    noun_count_r + "\nAvg sen length: " + ((sen_sum + 0d) / sen_count) +
    "\nSentences: " + sen_count + "\n");
89        }
90      }
91    }
92    catch (IOException e)
93    {
94      e.printStackTrace();
95    }
```

```
96    }
97
98    // Parse a line
99    private static void ParseLine(Matcher matcher, String pos)
100   {
101     ArrayList<String> sentence = new ArrayList<>();
102
103     while (matcher.find())
104     {
105       if (!matcher.group(2).equals("."))
106       {
107         sentence.add(matcher.group(2));
108       }
109       else
110       {
111         Collect(sentence, pos);
112       }
113     }
114     Collect(sentence, pos);
115   }
116
117   // Determine placement of terms and collect statistics
118   private static void Collect(ArrayList<String> sentence, String pos)
119   {
120     double mid, dist, sum = 0d, sum_l = 0d, sum_r = 0d;
121     int sen_length = sentence.size(), count = 0, count_l = 0, count_r =
        0;
122
123     if (sen_length > min_sen_thresh && sen_length < max_sen_thresh)
124     {
125       mid = (sen_length - 1d) / 2d;
126       for (int i = 0; i < sen_length; i++)
127       {
128         // NN - noun, VB - verb, JJ - adjective
129         if (sentence.get(i).contains(pos))
130         {
131           dist = (mid - i) / mid;
132           // i > mid means the term is on the right half
133           if (dist < 0)
134           {
135             sum_r += dist;
```

```
136               count_r++;
137             }
138             else if (dist > 0)
139             {
140               sum_l += dist;
141               count_l++;
142             }
143
144             sum += Math.abs(dist);
145             count++;
146           }
147         }
148
149         // Skip sentences with all the same pos
150         if (count != sen_length)
151         {
152           noun_sum += sum;
153           noun_sum_l += sum_l;
154           noun_sum_r += sum_r;
155           noun_count += count;
156           noun_count_l += count_l;
157           noun_count_r += count_r;
158           sen_sum += sen_length;
159           sen_count++;
160         }
161       }
162       sentence.clear();
163     }
164 }
```

Appendix B

Placement of Important Terms in Sentences Preliminary Experiment

B.1 Instructions

In order to run the Placement of Important Terms in Sentences preliminary experiment in Chapter 3.3, you need to do the following:

1. Copy and paste the Query Term Placement Parser code in Section B.2 into a file named RelParser.java.
2. Place the .jar file from Stanford Tagger 3.3.1 (stanford-postagger-3.3.1.jar) [68] into the same directory as RelParser.java.
3. Place the .jar file from H2 Database 1.3.174 (h2 -1.3.174.jar) [76] into the same directory as RelParser.java.
4. Copy the code of the Java implementation of the Porter Stemmer from [77] into RelParser.java.
5. Copmile the Query Term Placement Perser by entering "javac - cp .: stanford - postagger -3.3.1. jar : h2 -1.3.174. jar RelParser . java" into the command line.

6. Place the stopword_list.txt file from Terrier 3.5 [78] into the same directory as RelParser.java. Rename the file to stopwords.txt.

7. Place the english-left3words-distsim.tagger file from Stanford Tagger 3.3.1 [68] into the same directory as RelParser.java.

8. Place the collection directory into the same directory as RelParser.java. Rename the collection directory according to the collection as follows: WT2g to wt2g, disk4+5 to disk4+5, WT10g to wt10g, Blogs06 to blogs06, Gov2 to gov2.

9. Extract the archives in the collection in place. The contents of each archive should be extracted into the directory that contained the archive and the archive should be deleted. For example, if there is an archive B01.gz in the directory wt2g/WT01, after the extraction the directory wt2g/WT01 should contain the contents of B01.gz, B01, but not B01.gz.

10. Take all of the topics of the collection as defined in Section 5.1 and concatenate them into a single file named according to the collection. For example, topics.wt10g should contain the topics 451-550 for the WT10g collection. Place this file in the same directory as RelParser.java.

11. Take all of the query relevance files (.qrels) of the topics and concatenate them into a single file named according to the collection. For example, qrels.wt10g should contain the query relevance information for the topics 451-550 for the WT10g collection. Place this file in the same directory as RelParser.java.

12. Copy the properties specified in Appendix E into the terrier.properties file in the terrier-3.5/etc directory.

13. Generate the primary index as specified in Appendix F.

14. Enter "bin/trec_terrier.sh -r -Dtrec.model=BM25 -Dtrec.topics=collections/topics.[Data] -k1 1.2 -c [c]" into the command line. The [Data] and [c] parameters are mandatory. [Data] is the name of the collection directory, ex. wt2g. [c] is a tuning parameter for BM25. The values of [c] used in these experiments are as follows, WT2g = 0.2, WT10g = 0.3, disk4+5 = 0.3, Blogs06 = 0.2, Gov2 = 0.4. For example, for the WT10g, enter "bin/trec_terrier.sh -r -Dtrec.model=BM25 -Dtrec.topics=collections/topics.wt10g -k1 1.2 -c 0.3" into the command line.

15. Copy the .res file in the terrier-3.5/results directory into the same directory as RelParser.java. Rename the .res file according to the collection, ex. wt10g.res.

16. Run the Query Term Placement Parser by entering "java -cp .:stanford-postagger-3.3.1.jar:h2-1.3.174.jar RelParser [Data] [QL]" into the command line. Both the [Data] and [QL] parameters are mandatory. [Data] is the name of the collection directory, ex. wt2g. [QL] is a filter for the length of the query, only queries with length equal to [QL] will be parsed. [QL] = -1 means that all queries will be parsed. For example, to analyze the queries in the topics 451-550 with only 3 terms on the WT10g collection, enter "java -cp .:stanford-postagger-3.3.1.jar:h2-1.3.174.jar RelParser wt10g 3" into the command line.

The Query Term Placement Parser only supports the collections outlined in Section 5.1. Other collections are not guaranteed to work.

B.2 Query Term Placement Parser

```
1  // javac -cp .:stanford-postagger-3.3.1.jar:h2-1.3.174.jar RelParser.java
2  // java -cp .:stanford-postagger-3.3.1.jar:h2-1.3.174.jar RelParser data
       [query_length] [-i]
3
4  import java.io.BufferedReader;
5  import java.io.File;
6  import java.io.FileReader;
7  import java.io.IOException;
8  import java.sql.Connection;
9  import java.sql.DriverManager;
10 import java.sql.ResultSet;
11 import java.sql.SQLException;
12 import java.sql.Statement;
13 import java.util.ArrayList;
14 import java.util.HashMap;
15 import java.util.HashSet;
16 import java.util.regex.Matcher;
17 import java.util.regex.Pattern;
18 import edu.stanford.nlp.tagger.maxent.MaxentTagger;
19 import org.h2.Driver;
20
```

```
21 public class RelParser
22 {
23   private static MaxentTagger tagger = new
       MaxentTagger("english-left3words-distsim.tagger");
24   private static Statement statement = null;
25   private static Stemmer stemmer = new Stemmer();
26
27   // Parse output from tagger
28   private static Pattern pattern_tag =
       Pattern.compile("([^-\\s_]+)_([a-zA-Z\\.]+)");
29   private static HashSet<String> stopwords = new HashSet<>();
30
31   private static double r_term_sum = 0d, r_term_sum_l = 0d, r_term_sum_r
       = 0d, r_pos_norm = 0d, r_pos_norm_l = 0d, r_pos_norm_r = 0d,
       r_pos_log = 0d, r_pos_log_l = 0d, r_pos_log_r = 0d, r_score = 0d,
       r_score_l = 0d, r_score_r = 0d, r_sen_norm_sum = 0d, r_sen_log_sum =
       0d, nr_term_sum = 0d, nr_term_sum_l = 0d, nr_term_sum_r = 0d,
       nr_pos_norm = 0d, nr_pos_norm_l = 0d, nr_pos_norm_r = 0d, nr_pos_log
       = 0d, nr_pos_log_l = 0d, nr_pos_log_r = 0d, nr_score = 0d, nr_score_l
       = 0d, nr_score_r = 0d, nr_sen_norm_sum = 0d, nr_sen_log_sum = 0d;
32   private static int r_term_count = 0, r_term_count_l = 0,
       r_term_count_r = 0, r_pos = 0, r_pos_l = 0, r_pos_r = 0, r_sen_sum =
       0, r_sen_count = 0, nr_term_count = 0, nr_term_count_l = 0,
       nr_term_count_r = 0, nr_pos = 0, nr_pos_l = 0, nr_pos_r = 0,
       nr_sen_sum = 0, nr_sen_count = 0;
33   private static final double ln2 = Math.log(2);
34   private static boolean index;
35
36   private static final int min_sen_length = 7, max_sen_length = 20,
       min_sen_thresh = min_sen_length - 1, max_sen_thresh = max_sen_length
       + 1, min_line_thresh = min_sen_length * 2 - 2, max_line_thresh =
       max_sen_length * 100 + 1;
37
38   public static void main (String[] args)
39   {
40     Connection conn = null;
41
42     Matcher matcher;
43     // Extract topic query terms
44     Pattern pattern_topic =
       Pattern.compile("<num>\\s*Number:\\s*([0-9]+)[^<]*(</num>)?" +
```

```
45          "[^<]*<title>\\s*([^<]+)(</title>|<desc>)");
46      // Extract document ids from documents
47      Pattern pattern_id = Pattern.compile("<DOCNO>([^<]+)</DOCNO>");
48
49      HashMap<String, HashSet<String>> qrels = new HashMap<>(), topics =
        new HashMap<>();
50      // Map document ids to files
51      HashMap<String, String> paths = new HashMap<>();
52      // Delimiters for text in documents
53      HashMap<String, String> delims = new HashMap<>();
54
55      String[] split, pathSplit;
56      String file, path, text = "", line;
57      int query_length = (args.length > 1 ? Integer.parseInt(args[1]) :
        -1), rel = 0, nonrel = 0, files = 0, docs = 0;
58      boolean compilePath = args[0].equals("disk4+5");
59
60      delims.put("wt2g_start", "</DOCHDR>");
61      delims.put("wt2g_end", "</DOC>");
62      delims.put("disk4+5_start", "<TEXT>");
63      delims.put("disk4+5_end", "</TEXT>");
64      delims.put("wt10g_start", "</DOCHDR>");
65      delims.put("wt10g_end", "</DOC>");
66      delims.put("blogs06_start", "</DOCHDR>");
67      delims.put("blogs06_end", "</DOC>");
68      delims.put("gov2_start", "</DOCHDR>");
69      delims.put("gov2_end", "</DOC>");
70
71      index = args.length > 2 && args[2].equals("-i");
72
73      try
74      {
75        try (BufferedReader r = new BufferedReader(new
        FileReader("stopwords")))
76        {
77          while ((line = r.readLine()) != null)
78          {
79            stopwords.add(line);
80          }
81        }
82
```

```
83        // Read each file, map document ids to files
84        if (compilePath)
85        {
86          for (String d1 : (new File(args[0])).list())
87          {
88            for (String d2 : (new File(args[0] + "/" + d1)).list())
89            {
90              file = args[0] + "/" + d1 + "/" + d2;
91              try (BufferedReader r = new BufferedReader(new
    FileReader(file)))
92              {
93                while ((line = r.readLine()) != null)
94                {
95                  if (line.contains("<DOCNO>"))
96                  {
97                    matcher = pattern_id.matcher(line);
98                    if (matcher.find())
99                    {
100                     paths.put(matcher.group(1).trim(), file);
101                     docs++;
102                   }
103                 }
104               }
105             }
106             System.out.println("Files: " + ++files + " Docs: " + docs);
107           }
108         }
109       }
110
111       try (BufferedReader r = new BufferedReader(new
    FileReader("topics." + args[0])))
112       {
113         while ((line = r.readLine()) != null)
114         {
115           text += line;
116         }
117       }
118
119       matcher = pattern_topic.matcher(text);
120       while (matcher.find())
121       {
```

```
122         if (!topics.containsKey(matcher.group(1)))
123         {
124           topics.put(matcher.group(1), new HashSet<>());
125         }
126         // Remove stopwords and stem query terms
127         for (String s : matcher.group(3).toLowerCase().split("[^a-z]"))
128         {
129           if (s.length() > 1 && !stopwords.contains(s))
130           {
131             stemmer.add(s.toCharArray(), s.length());
132             stemmer.stem();
133             topics.get(matcher.group(1)).add(stemmer.toString());
134           }
135         }
136       }
137
138       try (BufferedReader r = new BufferedReader(new FileReader("qrels."
    + args[0])))
139       {
140         while ((line = r.readLine()) != null)
141         {
142           split = line.split("\\s");
143           // Add relevant documents to hashset
144           if (!split[3].equals("0"))
145           {
146             if (!qrels.containsKey(split[0]))
147             {
148               qrels.put(split[0], new HashSet<>());
149             }
150             qrels.get(split[0]).add(split[2]);
151           }
152         }
153       }
154
155       if (index)
156       {
157         conn = DriverManager.getConnection("jdbc:h2:" + args[0]);
158         statement = conn.createStatement();
```

```
159         statement.execute("CREATE TABLE IF NOT EXISTS temp (term
    VARCHAR(20), pos_l INT, pos_r INT, sen_length INT);" + "TRUNCATE
    TABLE temp;" + "CREATE INDEX IF NOT EXISTS temp_index ON temp(term);"
    + "CREATE TABLE IF NOT EXISTS index (doc VARCHAR(30), term
    VARCHAR(20), pos_l DOUBLE, pos_r DOUBLE, sen_length DOUBLE, PRIMARY
    KEY HASH (doc, term));" + "TRUNCATE TABLE index");
160       }
161
162       try (BufferedReader r = new BufferedReader(new FileReader(args[0]
    + ".res")))
163       {
164         while ((line = r.readLine()) != null)
165         {
166           split = line.split("\\s");
167
168           if (qrels.containsKey(split[0]) && (query_length == -1 ||
    query_length == topics.get(split[0]).size()))
169           {
170             if (compilePath)
171             {
172               path = paths.get(split[2]);
173             }
174             else
175             {
176               pathSplit = split[2].split("-");
177               if (args[0].equals("blogs06"))
178               {
179                 path = args[0] + "/" + pathSplit[1] + "/permalinks-" +
    pathSplit[2];
180               }
181               else
182               {
183                 path = args[0] + "/" + pathSplit[0] + "/" + pathSplit[1];
184               }
185             }
186
187             // A document is relevant if it's in qrels hashset
188             if (qrels.get(split[0]).contains(split[2]))
189             {
190               ReadDocument(path, split[2], delims.get(args[0] +
    "_start"), delims.get(args[0] + "_end"), topics.get(split[0]), true);
```

```
191                 rel++;
192               }
193               else
194               {
195                 ReadDocument(path, split[2], delims.get(args[0] +
"_start"), delims.get(args[0] + "_end"), topics.get(split[0]), false);
196                 nonrel++;
197               }
198
199               if (index)
200               {
201                 statement.execute("MERGE INTO index (SELECT '" + split[2]
+ "', term, AVG(0.0 + pos_l), AVG(0.0 + pos_r), AVG(0.0 + sen_length)
FROM temp GROUP BY term);" + "TRUNCATE TABLE temp");
202               }
203
204               System.out.println("\nRelevant docs: " + rel + "\nAvg dist:
" + (r_term_sum / r_term_count) + "\nAvg dist left: " + (r_term_sum_l
/ r_term_count_l) + "\nAvg dist right: " + (Math.abs(r_term_sum_r) /
r_term_count_r) + "\nAvg pos: " + ((r_pos + 0d) / r_term_count) +
"\nAvg pos left: " + ((r_pos_l + 0d) / r_term_count_l) + "\nAvg pos
right: " + ((r_pos_r + 0d) / r_term_count_r) + "\nAvg pos norm: " +
(r_pos_norm / r_term_count) + "\nAvg pos norm left: " + (r_pos_norm_l
/ r_term_count_l) + "\nAvg pos norm right: " + (r_pos_norm_r /
r_term_count_r) + "\nAvg pos log: " + (r_pos_log / r_term_count) +
"\nAvg pos log left: " + (r_pos_log_l / r_term_count_l) + "\nAvg pos
log right: " + (r_pos_log_r / r_term_count_r) + "\nScore: " +
(r_score / r_term_count) + "\nScore left: " + (r_score_l /
r_term_count_l) + "\nScore right: " + (r_score_r / r_term_count_r) +
"\nTerms: " + r_term_count + "\nTerms left: " + r_term_count_l +
"\nTerms right: " + r_term_count_r + "\nAvg sen length: " +
((r_sen_sum + 0d) / r_sen_count) + "\nAvg sen norm length: " +
(r_sen_norm_sum / r_sen_count) + "\nAvg sen log length: " +
(r_sen_log_sum / r_sen_count) + "\nSentences: " + r_sen_count + "\n");
```

```
205             System.out.println("Non-relevant docs: " + nonrel + "\nAvg
        dist: " + (nr_term_sum / nr_term_count) + "\nAvg dist left: " +
        (nr_term_sum_l / nr_term_count_l) + "\nAvg dist right: " +
        (Math.abs(nr_term_sum_r) / nr_term_count_r) + "\nAvg pos: " +
        ((nr_pos + 0d) / nr_term_count) + "\nAvg pos left: " + ((nr_pos_l +
        0d) / nr_term_count_l) + "\nAvg pos right: " + ((nr_pos_r + 0d) /
        nr_term_count_r) + "\nAvg pos norm: " + (nr_pos_norm / nr_term_count)
        + "\nAvg pos norm left: " + (nr_pos_norm_l / nr_term_count_l) +
        "\nAvg pos norm right: " + (nr_pos_norm_r / nr_term_count_r) + "\nAvg
        pos log: " + (nr_pos_log / nr_term_count) + "\nAvg pos log left: " +
        (nr_pos_log_l / nr_term_count_l) + "\nAvg pos log right: " +
        (nr_pos_log_r / nr_term_count_r) + "\nScore: " + (nr_score /
        nr_term_count) + "\nScore left: " + (nr_score_l / nr_term_count_l) +
        "\nScore right: " + (nr_score_r / nr_term_count_r) + "\nTerms: " +
        nr_term_count + "\nTerms left: " + nr_term_count_l + "\nTerms right:
        " + nr_term_count_r + "\nAvg sen length: " + ((nr_sen_sum + 0d) /
        nr_sen_count) + "\nAvg sen norm length: " + (nr_sen_norm_sum /
        nr_sen_count) + "\nAvg sen log length: " + (nr_sen_log_sum /
        nr_sen_count) + "\nSentences: " + nr_sen_count + "\n");
206           }
207         }
208       }
209
210       if (index)
211       {
212         statement.execute("DROP TABLE temp");
213         statement.close();
214         conn.close();
215       }
216     }
217     catch (IOException | SQLException e)
218     {
219       e.printStackTrace();
220     }
221   }
222
223   // Return text from document
224   private static void ReadDocument(String path, String id, String start,
        String end, HashSet<String> topic, boolean rel) throws SQLException
225   {
226     String line;
```

```
227     // Document ID found
228     boolean found = false;
229     // Start of document reached
230     boolean read = false;
231     // End of document reached
232     boolean stop = false;
233
234     try (BufferedReader r = new BufferedReader(new FileReader(path)))
235     {
236       while ((line = r.readLine()) != null && !stop)
237       {
238         if (read && !line.contains(end))
239         {
240           // Define end of line as end of sentence and parse
241           line =
    line.replaceAll("<[^>]*>|[^\u0020-\u002F\u003A-\u007F]", "") + " ! ";
242           if (line.length() > min_line_thresh && line.length() <
    max_line_thresh)
243           {
244             ParseLine(pattern_tag.matcher(tagger.tagString(line)),
    topic, rel);
245           }
246         }
247         else if (read)
248         {
249           stop = true;
250         }
251         else if (found && line.contains(start))
252         {
253           read = true;
254         }
255         else if (line.contains(id))
256         {
257           found = true;
258         }
259       }
260     }
261     catch (IOException e)
262     {
263       e.printStackTrace();
264     }
```

```
265   }
266
267   // Parse a line
268   private static void ParseLine(Matcher matcher, HashSet<String> topic,
        boolean rel) throws SQLException
269   {
270     ArrayList<String> sentence = new ArrayList<>();
271     String term;
272
273     while (matcher.find())
274     {
275       if (!matcher.group(2).equals("."))
276       {
277         term = matcher.group(1).toLowerCase();
278         sentence.add(term);
279       }
280
281       else
282       {
283         Collect(sentence, topic, rel);
284       }
285     }
286     Collect(sentence, topic, rel);
287   }
288
289   // Determine placement of terms and collect statistics
290   private static void Collect(ArrayList<String> sentence,
        HashSet<String> topic, boolean rel) throws SQLException
291   {
292     String term;
293     double mid, dist, sum = 0d, sum_l = 0d, sum_r = 0d, pos_norm = 0d,
      pos_norm_l = 0d, pos_norm_r = 0d, pos_log = 0d, pos_log_l = 0d,
      pos_log_r = 0d, score = 0d, score_l = 0d, score_r = 0d;
294     int sen_length = sentence.size(), dist_end, pos = 0, pos_l = 0,
      pos_r = 0, count = 0, count_l = 0, count_r = 0;
295
296     if (sen_length > min_sen_thresh && sen_length < max_sen_thresh)
297     {
298       mid = (sen_length - 1d) / 2d;
299       for (int i = 0; i < sen_length; i++)
300       {
```

```
301         stemmer.add(sentence.get(i).toCharArray(),
    sentence.get(i).length());
302         stemmer.stem();
303         term = stemmer.toString();
304         // If stemmed term matches a query term
305         if (topic.contains(term))
306         {
307           dist = (mid - i) / mid;
308
309           // i > mid means the term is on the right half
310           if (dist < 0)
311           {
312             dist_end = sen_length - 1 - i;
313             sum_r += dist;
314             pos_r += dist_end;
315             pos_norm_r += norm(dist_end);
316             pos_log_r += log(dist_end);
317             score_r += weight(mid - dist_end, sen_length, mid);
318             count_r++;
319
320             if (index)
321             {
322               statement.execute("INSERT INTO temp VALUES('" + term +
    "',NULL," + dist_end + "," + sen_length + ")");
323             }
324           }
325           else if (dist > 0)
326           {
327             sum_l += dist;
328             pos_l += i;
329             pos_norm_l += norm(i);
330             pos_log_l += log(i);
331             score_l += weight(mid - i, sen_length, mid);
332             count_l++;
333
334             if (index)
335             {
336               statement.execute("INSERT INTO temp VALUES('" + term +
    "'," + i + ",NULL," + sen_length + ")");
337             }
338           }
```

```
339
340             sum += Math.abs(dist);
341             pos += i;
342             pos_norm += norm(i);
343             pos_log += log(i);
344             score += weight(Math.abs(mid - i), sen_length, mid);
345             count++;
346           }
347         }
348
349         // Skip sentences with no query terms
350         if (count != 0)
351         {
352           if (rel)
353           {
354             r_term_sum += sum;
355             r_term_sum_l += sum_l;
356             r_term_sum_r += sum_r;
357             r_pos += pos;
358             r_pos_l += pos_l;
359             r_pos_r += pos_r;
360             r_pos_norm += pos_norm;
361             r_pos_norm_l += pos_norm_l;
362             r_pos_norm_r += pos_norm_r;
363             r_pos_log += pos_log;
364             r_pos_log_l += pos_log_l;
365             r_pos_log_r += pos_log_r;
366             r_score += score;
367             r_score_l += score_l;
368             r_score_r += score_r;
369             r_term_count += count;
370             r_term_count_l += count_l;
371             r_term_count_r += count_r;
372             r_sen_sum += sen_length;
373             r_sen_norm_sum += norm(sen_length);
374             r_sen_log_sum += log(sen_length);
375             r_sen_count++;
376           }
377           else
378           {
379             nr_term_sum += sum;
```

```
380             nr_term_sum_l += sum_l;
381             nr_term_sum_r += sum_r;
382             nr_pos += pos;
383             nr_pos_l += pos_l;
384             nr_pos_r += pos_r;
385             nr_pos_norm += pos_norm;
386             nr_pos_norm_l += pos_norm_l;
387             nr_pos_norm_r += pos_norm_r;
388             nr_pos_log += pos_log;
389             nr_pos_log_l += pos_log_l;
390             nr_pos_log_r += pos_log_r;
391             nr_score += score;
392             nr_score_l += score_l;
393             nr_score_r += score_r;
394             nr_term_count += count;
395             nr_term_count_l += count_l;
396             nr_term_count_r += count_r;
397             nr_sen_sum += sen_length;
398             nr_sen_norm_sum += norm(sen_length);
399             nr_sen_log_sum += log(sen_length);
400             nr_sen_count++;
401           }
402         }
403       }
404       sentence.clear();
405     }
406
407     // Normalize the value
408     private static double norm(int value)
409     {
410       return value / (1d + value);
411     }
412
413     // Calculate the base 2 log of a value
414     private static double log(double value)
415     {
416       return Math.log(1d + value) / ln2;
417     }
418
419     private static double weight(double dist, int sen_length, double mid)
420     {
```

```
421       double thresh = mid - sen_length / 3d + 2d;
422       return (dist >= thresh ? 1d : 1d - Math.exp(Math.pow(dist, 2) / (-2d
          * Math.pow(thresh, 2))));
423     }
424   }
```

Appendix C

Effectiveness of Proposed Weighting Method Preliminary Experiment

C.1 Instructions

In order to run the Effectiveness of Proposed Weighting Method preliminary experiment in Chapter 3.4, you need to do the following:

1. Copy and paste the Weighting Method Parser code in Section C.2 into a file named TextParser.java.
2. Place the .jar file from Stanford Tagger 3.3.1 (stanford-postagger-3.3.1.jar) [68] into the same directory as TextParser.java.
3. Copy the code of the Java implementation of the Porter Stemmer from [77] into TextParser.java.
4. Compile the Noun Placement Parser by entering "javac -cp .:stanford-postagger-3.3.1.jar TextParser.java" into the command line.
5. Place the english-left3words-distsim.tagger file from Stanford Tagger 3.3.1 [68] into the same directory as TextParser.java.

6. Place the plain text document to analyze into the same directory as TextParser.java.

7. Run the Weighting Method Parser by entering "java -cp .:stanford-postagger-3.3.1.jar TextParser [Doc] [Denom] -compact" into the command line. The [Doc] and [Denom] parameters are mandatory. [Doc] is the name of the plain text document to analyze. [Denom] controls the width of the sentence segments, where a larger [Denom] means the middle sentence segment will be larger. [Denom] should be adjusted such that *Balance* is as close to 0 as possible. The Weighting Method Parser will print *Balance* out to the console to facilitate this. For example, to analyze a file named "hardtimes.txt", enter "java -cp .:stanford-postagger-3.3.1.jar TextParser hardtimes.txt 3" into the command line.

C.2 Weighting Method Parser

```
1  // javac -cp .:stanford-postagger-3.3.1.jar TextParser.java
2  // java -cp .:stanford-postagger-3.3.1.jar TextParser doc denom
       [-compact] [-table columns (default: 4) score_thresh (default: 5)]
3
4  import java.io.BufferedReader;
5  import java.io.FileReader;
6  import java.io.IOException;
7  import java.io.PrintWriter;
8  import java.util.ArrayList;
9  import java.util.Comparator;
10 import java.util.HashMap;
11 import java.util.HashSet;
12 import java.util.Map;
13 import java.util.regex.Matcher;
14 import java.util.regex.Pattern;
15 import java.util.TreeMap;
16 import edu.stanford.nlp.tagger.maxent.MaxentTagger;
17
18 public class TextParser
19 {
20   private static Stemmer stemmer = new Stemmer();
21
22   private static HashMap<String, Integer> results = new HashMap<>();
23   private static TreeMap<String, Integer> results_sorted = new
       TreeMap<>(new ValueComparator(results));
```

```
24    private static HashSet<String> stopwords = new HashSet<>();
25
26    private static final int min_sen_length = 7, max_sen_length = 20,
        min_sen_thresh = min_sen_length - 1, max_sen_thresh = max_sen_length
        + 1, min_line_thresh = min_sen_length * 2 - 2, max_line_thresh =
        max_sen_length * 100 + 1;
27
28    public static void main (String[] args)
29    {
30      MaxentTagger tagger = new
        MaxentTagger("english-left3words-distsim.tagger");
31      // Parse output from tagger
32      Pattern pattern_tag = Pattern.compile("([^-\\s_]+)_([a-zA-Z\\.]+)");
33
34      String line;
35      // The denominator for splitting the sentence
36      double denom = Double.parseDouble(args[1]);
37      int columns = 3, score_thresh = 4, balance = 0, i = 0;
38      boolean compact = false, table = false;
39
40      for (i = 2; i < args.length; i++)
41      {
42        switch (args[i])
43        {
44          // Print only nouns and non-stopwords
45          case "-compact":
46            compact = true;
47            break;
48          // Print results in the format of a multi-columned latex table
49          case "-table":
50            table = true;
51            if (args.length > ++i)
52            {
53              columns = Integer.parseInt(args[i]) - 1;
54            }
55            if (args.length > ++i)
56            {
57              score_thresh = Integer.parseInt(args[i]) - 1;
58            }
59            break;
60        }
```

```
61      }
62
63      try
64      {
65        try (BufferedReader r = new BufferedReader(new
        FileReader("stopwords")))
66        {
67          while ((line = r.readLine()) != null)
68          {
69            stopwords.add(line);
70          }
71        }
72
73        try (BufferedReader r = new BufferedReader(new
        FileReader(args[0])))
74        {
75          while ((line = r.readLine()) != null)
76          {
77            // Define end of line as end of sentence and parse
78            line = line.replace("[^\u0020-\u002F\u003A-\u007F]", "") + " !
        ";
79            ParseLine(pattern_tag.matcher(tagger.tagString(line)), denom,
        compact);
80          }
81        }
82
83        results_sorted.putAll(results);
84        try (PrintWriter w = new PrintWriter("out", "UTF-8"))
85        {
86          if (table)
87          {
88            i = 0;
89            //For outputting results in Latex table format
90            for (Map.Entry<String, Integer> e : results_sorted.entrySet())
91            {
92              if (Math.abs(e.getValue()) > score_thresh)
93              {
94                w.write(e.getKey() + " & " + e.getValue());
95                if (i != columns)
96                {
97                  w.write(" & ");
```

```
98                  i++;
99                }
100               else
101               {
102                 w.write(" \\\\ \\hline\n");
103                 i = 0;
104               }
105             }
106           }
107         }
108
109         else
110         {
111           for (Map.Entry<String, Integer> e : results_sorted.entrySet())
112           {
113             balance += e.getValue();
114             w.write(e.getKey() + " - " + e.getValue() + "\n");
115           }
116           w.write("\nBalance: " + balance + "\n");
117           System.out.println("Balance: " + balance);
118         }
119       }
120     }
121     catch (IOException e)
122     {
123       e.printStackTrace();
124     }
125   }
126
127   // Parse a line
128   private static void ParseLine(Matcher matcher, double denom, boolean
      compact)
129   {
130     ArrayList<String> sentence = new ArrayList<>(), sentence_pos = new
      ArrayList<>();
131     String pos;
132
133     while (matcher.find())
134     {
135       pos = matcher.group(2);
136       if (!pos.equals("."))
```

```
137         {
138           sentence.add(matcher.group(1).toLowerCase());
139           sentence_pos.add(pos);
140         }
141         else
142         {
143           Collect(sentence, sentence_pos, denom, compact);
144         }
145       }
146       Collect(sentence, sentence_pos, denom, compact);
147     }
148
149     // Determine placement of terms and assign scores
150     private static void Collect(ArrayList<String> sentence,
        ArrayList<String> sentence_pos, double denom, boolean compact)
151     {
152       String term;
153       double dist_max;
154       int sen_length = sentence.size(), score;
155
156       if (sen_length > min_sen_thresh && sen_length < max_sen_thresh)
157       {
158         dist_max = sen_length / denom;
159         for (int i = 0; i < sen_length; i++)
160         {
161           term = sentence.get(i);
162           if (!compact || (sentence_pos.get(i).contains("NN") &&
        !stopwords.contains(term)))
163           {
164             stemmer.add(term.toCharArray(), term.length());
165             stemmer.stem();
166             term = stemmer.toString();
167
168             // Score is 1 if term is in the first or last part of the
        sentence
169             score = (i < dist_max  || i > sen_length - dist_max - 1 ? 1 :
        -1);
170             if (results.containsKey(term))
171             {
172               score += results.get(term);
173             }
```

```
174             results.put(term, score);
175           }
176         }
177       }
178       sentence.clear();
179       sentence_pos.clear();
180     }
181 }
182
183 // For sorting results by descending score
184 class ValueComparator implements Comparator<String>
185 {
186   HashMap<String, Integer> base;
187
188   public ValueComparator(HashMap<String, Integer> base)
189   {
190     this.base = base;
191   }
192
193   public int compare(String a, String b)
194   {
195     return (base.get(a) >= base.get(b) ? -1 : 1);
196   }
197 }
```

C.3 Complete Results

The complete results from the Effectiveness of Proposed Weighting Method preliminary experiment in Chapter 3.4 is listed below. A noun n is shown if $|Score(n)| \geq 5$:

Term	Score	Term	Score	Term	Score	Term	Score
mr.	86	louisa	43	mrs.	31	gradgrind	27
tom	23	bounderbi	22	father	18	slackbridg	15
wai	15	miss	15	coketown	13	ma'am	13
man	12	friend	11	manner	11	wi	11
ti	11	stephen	11	sissi	11	thoma	9
creatur	9	morn	9	face	9	rachael	9

Continued on next page

Term	Score	Term	Score	Term	Score	Term	Score
daughter	8	death	8	chanc	8	wind	8
purpos	8	jame	8	women	8	machineri	7
action	7	sofa	7	child	7	’em	7
work	7	dai	7	fellow	7	anim	7
room	7	associ	6	consequ	6	hope	6
combin	6	merryleg	6	minut	6	faith	6
‘’	6	town	6	lodg	6	influenc	6
term	6	part	6	night	6	fortun	6
afternoon	5	mistak	5	number	5	retreat	5
ith	5	materi	5	bottom	5	pride	5
hour	5	natur	5	walk	5	o’clock	5
degre	5	contrari	5	justic	5	board	5
girl	5	new	5	famili	5	fairi	5
staircas	5	convers	5	pound	5	compani	5
dread	5	cheek	5	rope	5	weather	-5
sight	-5	idea	-5	world	-5	parti	-5
sundai	-5	shop	-5	fire	-5	heart	-5
advantag	-5	wish	-5	nowt	-5	porter	-5
visit	-5	nod	-5	principl	-5	uth	-5
babi	-5	journei	-5	spirit	-5	half	-5
children	-6	silenc	-6	horth	-6	deal	-6
letter	-6	master	-6	home	-6	pit	-6
fit	-6	whole	-6	men	-6	length	-6
build	-6	husband	-6	distanc	-6	tip	-6
case	-7	system	-7	church	-7	chair	-7
hat	-7	effect	-7	shake	-7	door	-7
blackpool	-8	reason	-8	period	-8	side	-9
corner	-9	whelp	-9	sleari	-9	voic	-9
window	-9	mind	-9	gentleman	-10	peopl	-10
moment	-10	mean	-11	boi	-13	sparsit	-13
ey	-13	bank	-13	time	-16	countri	-16
hand	-16	ladi	-17	head	-24		

Appendix D

Modifying Terrier

D.1 Instructions

Terrier 3.5 [78] was used in the experiments in Chapter 6. The code changes to terrier are listed in Section D.2. The code changes are obtained by running the diff command on Ubuntu 12.04 x86 between the modified and unmodified Terrier 3.5 directories. In order for the changes to take effect, Terrier needs to be recompiled. This can be done by entering "ant" in the command line in the terrier-3.5 directory.

D.2 Code Changes

terrier-3.5/src/core/org/terrier/applications/desktop/DesktopTerrier.java

```
1002c1002
<         queryingManager.runMatching(srq);
---
>         queryingManager.runMatching(srq, null, null, 1.2, 0, 0, 0, 0, 0);
```

terrier-3.5/src/core/org/terrier/applications/InteractiveQuerying.java

```
144c144
<       queryingManager.runMatching(srq);
---
>       queryingManager.runMatching(srq, null, null, 1.2, 0, 0, 0, 0, 0);
```

terrier-3.5/src/core/org/terrier/applications/TRECQueryingExpansion.java

```
87c87
<      queryingManager.runMatching(srq);
---
>      queryingManager.runMatching(srq, null, null, 1.2, 0, 0, 0, 0, 0);
```

terrier-3.5/src/core/org/terrier/applications/TRECQuerying.java

```
38a39,42
> import java.sql.Connection;
> import java.sql.DriverManager;
> import java.sql.SQLException;
> import java.sql.Statement;
156c160,163
<
---
>
>    Connection conn = null, conn1 = null;
>    Statement statement = null, statement1 = null;
>
624c631
<      return processQuery(queryId, query, 1.0, false);
---
>      return processQuery(queryId, query, 1.0, false, 1.2, 0, 0, 0, 0,
    0);
640c647
<      return processQuery(queryId, query, cParameter, true);
---
>      return processQuery(queryId, query, cParameter, true, 1.2, 0, 0,
    0, 0, 0);
658,659c665,666
<        double cParameter, boolean c_set) {
<      SearchRequest srq = processQuery(queryId, query, cParameter,
    c_set);
---
>        double cParameter, boolean c_set, double k_1, double d, double
    e, double f, double a, int kernel) {
>      SearchRequest srq = processQuery(queryId, query, cParameter,
    c_set, k_1, d, e, f, a, kernel);
694c701
<        double cParameter, boolean c_set) {
---
```

```
30 >       double cParameter, boolean c_set, double k_1, double d, double
       e, double f, double a, int kernel) {
31 724c731
32 <
33 ---
34 >
35 735c742
36 <     queryingManager.runMatching(srq);
37 ---
38 >     queryingManager.runMatching(srq, statement, statement1, k_1, d, e,
       f, a, kernel);
39 760c767
40 <     return processQueries(1.0d, false);
41 ---
42 >     return processQueries(1.0d, false, 1.2, 0, 0, 0, 0, 0, 0);
43 776c783
44 <     return processQueries(c, true);
45 ---
46 >     return processQueries(c, true, 1.2, 0, 0, 0, 0, 0, 0);
47 827c834
48 <   public String processQueries(double c, boolean c_set) {
49 ---
50 >   public String processQueries(double c, boolean c_set, double k_1,
       double d, double e, double f, double a, int kernel, int fold) {
51 837a845,869
52 >     try
53 >     {
54 >       Class.forName("org.h2.Driver");
55 >     }
56 >     catch (ClassNotFoundException ex)
57 >     {
58 >       ex.printStackTrace();
59 >     }
60 >
61 >     try
62 >     {
63 >       conn = DriverManager.getConnection("jdbc:h2:./var/index/index");
64 >       statement = conn.createStatement();
65 >
66 >       /*conn1 = DriverManager.getConnection("jdbc:h2:var/index/cache");
67 >       statement1 = conn1.createStatement();
```

```
68  >
69  >       statement1.execute("create table if not exists index (doc
    varchar(15), term varchar(20), dist double, length double, avg_length
    double, primary key hash (doc, " +
70  >               "" + "term"));" + "truncate table index");*/
71  >     }
72  >     catch (SQLException ex)
73  >     {
74  >       ex.printStackTrace();
75  >     }
76  >
77  842,850c874,885
78  <       // process the query
79  <       long processingStart = System.currentTimeMillis();
80  <       processQueryAndWrite(qid, query, c, c_set);
81  <       long processingEnd = System.currentTimeMillis();
82  <       if (logger.isInfoEnabled())
83  <         logger
84  <             .info("Time to process query: "
85  <                 + ((processingEnd - processingStart) / 1000.0D));
86  <       doneSomeTopics = true;
87  ---
88  >
89  >       if (fold == 0 || Integer.parseInt(qid) % 2 == fold % 2) {
90  >         // process the query
91  >         long processingStart = System.currentTimeMillis();
92  >         processQueryAndWrite(qid, query, c, c_set, k_1, d, e, f, a,
    kernel);
93  >         long processingEnd = System.currentTimeMillis();
94  >         if (logger.isInfoEnabled())
95  >           logger
96  >                   .info("Time to process query: "
97  >                           + ((processingEnd - processingStart) /
    1000.0D));
98  >         doneSomeTopics = true;
99  >       }
100 869a905,918
101 >
102 >     try
103 >     {
104 >       statement.close();
```

```
105 >         conn.close();
106 >
107 >         /*statement1.close();
108 >         conn1.close();*/
109 >       }
110 >       catch (SQLException ex)
111 >       {
112 >         ex.printStackTrace();
113 >       }
114 >
115 1089a1139,1149
116 >       }
117 >     }
118 >
119 >     public class Term
120 >     {
121 >       byte pos, length;
122 >
123 >       public Term(int pos, int length)
124 >       {
125 >         this.pos = (byte)pos;
126 >         this.length = (byte)length;
```

terrier-3.5/src/core/org/terrier/applications/TrecTerrier.java

```
1  159a160,161
2  >     protected double k_1 = 1.2, d = 0, e = 0, f = 0, a = 0;
3  >     protected int kernel, fold;
4  317a320,333
5  >         else if (args[pos].startsWith("-k1"))
6  >           k_1 = Double.parseDouble(args[++pos]);
7  >         else if (args[pos].startsWith("-dd"))
8  >           d = Double.parseDouble(args[++pos]);
9  >         else if (args[pos].startsWith("-ee"))
10 >           e = Double.parseDouble(args[++pos]);
11 >         else if (args[pos].startsWith("-ff"))
12 >           f = Double.parseDouble(args[++pos]);
13 >         else if (args[pos].startsWith("-aa"))
14 >           a = Double.parseDouble(args[++pos]);
15 >             else if (args[pos].startsWith("-kk"))
16 >               kernel = Integer.parseInt(args[++pos]);
17 >             else if (args[pos].startsWith("-fold"))
```

```
18 >             fold = Integer.parseInt(args[++pos]);
19 394c410
20 <       trecQuerying.processQueries(c, isParameterValueSpecified);
21 ---
22 >       trecQuerying.processQueries(c, isParameterValueSpecified, k_1,
       d, e, f, a, kernel, fold);
```

terrier-3.5/src/core/org/terrier/indexing/BasicIndexer.java

```
1  30a31,35
2  > import java.sql.Connection;
3  > import java.sql.DriverManager;
4  > import java.sql.SQLException;
5  > import java.sql.Statement;
6  > import java.util.ArrayList;
7  85c90
8  <     public void processTerm(String term)
9  ---
10 >     public void processTerm(String term, Statement statement, int pos,
       int length)
11 91c96
12 <         termsInDocument.insert(term);
13 ---
14 >         termsInDocument.insert(term, null, 0, 0);
15 109c114
16 <     public void processTerm(String term)
17 ---
18 >     public void processTerm(String term, Statement statement, int pos,
       int length)
19 193,194c198,211
20 <    * @param collections Collection[] the collections to be indexed.
21 <    */
22 ---
23 >    */
24 >
25 >   void ProcessSentence(ArrayList<String> sentence, Statement statement)
26 >   {
27 >     int length = sentence.size();
28 >     for (int i = 0; i < length; i++)
29 >     {
30 >       //termFields = doc.getFields();
31 >       /* pass term into TermPipeline (stop, stem etc) */
```

```
32 >       pipeline_first.processTerm(sentence.get(i), statement, i,
   length);
33 >       /* the term pipeline will eventually add the term to this
   object. */
34 >     }
35 >     sentence.clear();
36 >   }
37 213c230,247
38 <
39 ---
40 >
41 >     /*Connection conn = null;
42 >     Statement statement = null;
43 >     try
44 >     {
45 >       conn = DriverManager.getConnection("jdbc:h2:var/index/index");
46 >       statement = conn.createStatement();
47 >       statement.execute("create table temp (term varchar(20), dist
   double, length int);" + "create index temp_index on temp(term);" +
   "create table index (doc varchar(15), " +
48 >               "" + "term varchar(20), dist double, length double,
   primary key hash (doc, term));" + "create table meta (doc varchar(15)
   primary key hash, avg_length double)");
49 >     }
50 >     catch (SQLException e)
51 >     {
52 >       e.printStackTrace();
53 >     }
54 >
55 >     ArrayList<String> sentence = new ArrayList<String>();
56 >     String docno;*/
57 >
58 235c269
59 <
60 ---
61 >
62 241c275,279
63 <
64 ---
65 >
66 >         /*docno = "";
```

```
67 >          for (String s : doc.getProperty("docno").split("-"))
68 >            docno+= s.trim();*/
69 >
70 244,249c282,287
71 <            if ((term = doc.getNextTerm())!=null && !term.equals("")) {
72 <              termFields = doc.getFields();
73 <              /* pass term into TermPipeline (stop, stem etc) */
74 <              pipeline_first.processTerm(term);
75 <              /* the term pipeline will eventually add the term to this
       object. */
76 <            }
77 ---
78 >            if ((term = doc.getNextTerm()) != null)
79 >              pipeline_first.processTerm(term, null, 0, 0);
80 >            /*if (term.equals(""))
81 >                ProcessSentence(sentence, statement);
82 >            else
83 >              sentence.add(term);*/
84 253a292,295
85 >
86 >          /*if (sentence.size() > 0)
87 >            ProcessSentence(sentence, statement);*/
88 >
89 258a301,310
90 >          /*try
91 >          {
92 >            statement.execute("insert into index (select '" + docno +
       "', term, avg(dist), avg(length) from temp group by term);" + "insert
       into meta (select '" + docno + "', " +
93 >                    "" + "avg(length) from temp);"  + "truncate table
       temp");
94 >          }
95 >          catch (SQLException e)
96 >          {
97 >            e.printStackTrace();
98 >          }*/
99 >
100 362a415,424
101 >      /*try
102 >      {
103 >        statement.execute("drop table temp");
```

```
104 >         statement.close();
105 >         conn.close();
106 >       }
107 >       catch (SQLException e)
108 >       {
109 >         e.printStackTrace();
110 >       }*/
```

terrier-3.5/src/core/org/terrier/indexing/BasicSinglePassIndexer.java

```
1  216c216
2  <               pipeline_first.processTerm(term);
3  ---
4  >               pipeline_first.processTerm(term, null, 0, 0);
```

terrier-3.5/src/core/org/terrier/indexing/BlockIndexer.java

```
1  32a33
2  > import java.sql.Statement;
3  92c93
4  <       public void processTerm(String t) {
5  ---
6  >       public void processTerm(String t, Statement statement, int pos,
       int length) {
7  118c119
8  <       public void processTerm(String t) {
9  ---
10 >       public void processTerm(String t, Statement statement, int pos,
       int length) {
11 168c169
12 <       public void processTerm(String t) {
13 ---
14 >       public void processTerm(String t, Statement statement, int pos,
       int length) {
15 213c214
16 <       public void processTerm(String t) {
17 ---
18 >       public void processTerm(String t, Statement statement, int pos,
       int length) {
19 375c376
20 <               pipeline_first.processTerm(term);
21 ---
22 >               pipeline_first.processTerm(term, null, 0, 0);
```

terrier-3.5/src/core/org/terrier/indexing/BlockSinglePassIndexer.java

```
35a36
> import java.sql.Statement;
67c68
<     public void processTerm(String t) {
---
>     public void processTerm(String t, Statement statement, int pos,
    int length) {
93c94
<     public void processTerm(String t) {
---
>     public void processTerm(String t, Statement statement, int pos,
    int length) {
143c144
<     public void processTerm(String t) {
---
>     public void processTerm(String t, Statement statement, int pos,
    int length) {
188c189
<     public void processTerm(String t) {
---
>     public void processTerm(String t, Statement statement, int pos,
    int length) {
```

terrier-3.5/src/core/org/terrier/indexing/ExtensibleSinglePassIndexer.java

```
153c153
<             pipeline_first.processTerm(term);
---
>             pipeline_first.processTerm(term, null, 0, 0);
```

terrier-3.5/src/core/org/terrier/indexing/tokenisation/EnglishTokeniser.java

```
79a80
>     boolean skip = false;
97c98,101
<         ch = this.br.read();
---
>         /*if (!skip)
>         {
```

```
8  >            ch = this.br.read();
9  >          }*/
10 101,102c105
11 <            while (ch != -1 && (ch < 'A' || ch > 'Z') && (ch < 'a' || ch
       > 'z')
12 <              && (ch < '0' || ch > '9')
13 ---
14 >            while (ch != -1 && (ch < 'A' || ch > 'Z') && (ch < 'a' || ch
       > 'z') && (ch < '0' || ch > '9') //ch != '!' && ch != '?' && ch !=
       '.' && ch != ';' && ch != '\n' && ch != '\r' &&
15 110,114c113,114
16 <            //now accept all alphanumeric charaters
17 <            while (ch != -1 && (
18 <              ((ch >= 'A') && (ch <= 'Z'))
19 <              || ((ch >= 'a') && (ch <= 'z'))
20 <              || ((ch >= '0') && (ch <= '9'))))
21 ---
22 >
23 >            if (ch == '!' || ch == '?' || ch == '.' || ch == ';')// ||
       ch == '\n' || ch == '\r')
24 116,123c116,119
25 <              /* add character to word so far */
26 <              sw.append((char)ch);
27 <              ch = br.read();
28 <              counter++;
29 <            }
30 <            if (sw.length() > MAX_TERM_LENGTH)
31 <              if (DROP_LONG_TOKENS)
32 <                return null;
33 ---
34 >              /*if (skip)
35 >              {
36 >                skip = false;
37 >              }
38 125,128c121,150
39 <                sw.setLength(MAX_TERM_LENGTH);
40 <            String s = check(sw.toString());
41 <            if (s.length() > 0)
42 <              return s;
43 ---
44 >              {*/
```

```
45 >                counter++;
46 >              //}
47 >              return "";
48 >            }
49 >            else
50 >            {
51 >              //now accept all alphanumeric charaters
52 >              while (ch != -1 && ((ch >= 'A') && (ch <= 'Z')) || ((ch >=
    'a') && (ch <= 'z')) || ((ch >= '0') && (ch <= '9')))
53 >              {
54 >                /* add character to word so far */
55 >                sw.append((char)ch);
56 >                ch = br.read();
57 >                counter++;
58 >              }
59 >
60 >              /*if (ch == '!' || ch == '?' || ch == '.' || ch == ';' ||
    ch == '\n' || ch == '\r')
61 >              {
62 >                skip = true;
63 >              }*/
64 >
65 >              if (sw.length() > MAX_TERM_LENGTH)
66 >                if (DROP_LONG_TOKENS)
67 >                  return null;
68 >                else
69 >                  sw.setLength(MAX_TERM_LENGTH);
70 >              String s = check(sw.toString());
71 >              if (s.length() > 0)
72 >                return s;
73 >            }
```

terrier-3.5/src/core/org/terrier/indexing/TRECFullTokenizer.java

```
1  455,460c455,462
2  <          (!hasWhitelist || (hasWhitelist && inTagToProcess )) &&
3  <          !inTagToSkip)
4  <      {
5  <        if (!stk.empty() && exactTagSet.isTagToProcess(stk.peek()))
6  <          return lowercase ? s.toLowerCase() : s;
7  <        //}
8  ---
```

```
9  >          (!hasWhitelist || (hasWhitelist && inTagToProcess )) &&
10 >          !inTagToSkip)
11 >      {
12 >        if (!stk.empty() && tagSet.isIdTag(stk.peek()))
13 >          return s;
14 >        if (!stk.empty() && exactTagSet.isTagToProcess(stk.peek()))
15 >          return lowercase ? s.toLowerCase() : s;
16 >        //}
```

terrier-3.5/src/core/org/terrier/matching/AccumulatorResultSet.java

```
1  126a127,137
2  >    public void Sort()
3  >    {
4  >      this.docids = scoresMap.keys();
5  >      this.scores = scoresMap.getValues();
6  >      this.occurrences = occurrencesMap.getValues();
7  >
8  >      this.arraysInitialised = true;
9  >
10 >      HeapSort.descendingHeapSort(this.getScores(), this.getDocids(),
       this.getOccurrences(), this.docids.length);
11 >    }
12 >
```

terrier-3.5/src/core/org/terrier/matching/BaseMatching.java

```
1  30a31
2  > import java.sql.Statement;
3  35d35
4  <
5  37d36
6  <
7  49d47
8  <
9  322c320,321
10 <    public abstract ResultSet match(String queryNumber,
       MatchingQueryTerms queryTerms) throws IOException;
11 ---
12 >    public abstract ResultSet match(String queryNumber,
       MatchingQueryTerms queryTerms, Statement statement, Statement
       statement1, double k_1, double d, double e, double f, double a,
13 >                                    int kernel) throws IOException;
```

terrier-3.5/src/core/org/terrier/matching/daat/Full.java

```
33a34
> import java.sql.Statement;
77c78,79
<   public ResultSet match(String queryNumber, MatchingQueryTerms
    queryTerms) throws IOException
---
>   public ResultSet match(String queryNumber, MatchingQueryTerms
    queryTerms, Statement statement, Statement statement1, double k_1,
    double d, double e, double f, double a,
>                          int kernel) throws IOException
81c83
<     plm = new PostingListManager(index, super.collectionStatistics,
    queryTerms);
---
>     plm = new PostingListManager(index, super.collectionStatistics,
    queryTerms, true);
167c169
<         cc.updateScore(plm.score(i));
---
>         cc.updateScore(plm.score(i, 1.2));
```

terrier-3.5/src/core/org/terrier/matching/daat/FullNoPLM.java

```
33a34
> import java.sql.Statement;
68c69,70
<   public ResultSet match(String queryNumber, MatchingQueryTerms
    queryTerms) throws IOException
---
>   public ResultSet match(String queryNumber, MatchingQueryTerms
    queryTerms, Statement statement, Statement statement1, double k_1,
    double d, double e, double f, double a,
>                          int kernel) throws IOException
```

terrier-3.5/src/core/org/terrier/matching/dsms/DependenceScoreModifier.java

```
144c144
<       PostingListManager plm = new PostingListManager(index,
    index.getCollectionStatistics(), terms);
```

```
3  ---
4  >        PostingListManager plm = new PostingListManager(index,
       index.getCollectionStatistics(), terms, true);
```

terrier-3.5/src/core/org/terrier/matching/Matching.java

```
1  29a30
2  > import java.sql.Statement;
3  48c49,50
4  <    ResultSet match(String queryNumber, MatchingQueryTerms queryTerms)
       throws IOException;
5  ---
6  >    ResultSet match(String queryNumber, MatchingQueryTerms queryTerms,
       Statement statement, Statement statement1, double k_1, double d,
       double e, double f, double a,
7  >                    int kernel) throws IOException;
```

terrier-3.5/src/core/org/terrier/matching/models/BB2.java

```
1  28a29,31
2  >
3  > import java.sql.Statement;
4  >
5  122a126,131
6  >    }
7  >
8  >    @Override
9  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
10 >      // TODO Auto-generated method stub
11 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/BM25.java

```
1  28a29,31
2  >
3  > import java.sql.*;
4  >
5  36c39
6  <    */
7  ---
8  >   */
9  38c41,230
```

```
10  <    private static final long serialVersionUID = 1L;
11  ---
12  >    private static final long serialVersionUID = 1L;
13  >
14  >    /** The constant k_3.*/
15  >    private double k_3=8d;
16  >
17  >    /** The parameter b.*/
18  >    private double b;
19  >
20  >    /** A default constructor.*/
21  >    public BM25() {
22  >      super();
23  >      k_1 = 1.2d;
24  >      b=0.75d;
25  >    }
26  >    /**
27  >     * Returns the name of the model.
28  >     * @return the name of the model
29  >     */
30  >    public final String getInfo() {
31  >      return "BM25b"+b;
32  >    }
33  >    /**
34  >     * Uses BM25 to compute a weight for a term in a document.
35  >     * @param tf The term frequency in the document
36  >     * @param docLength the document's length
37  >     * @return the score assigned to a document with the given
38  >     *         tf and docLength, and other preset parameters
39  >     */
40  >    public double score(double tf, double docLength) {
41  >      double K = k_1 * ((1 - b) + b * docLength / averageDocumentLength)
        + tf;
42  >      return (tf * (k_3 + 1d) * keyFrequency / ((k_3 + keyFrequency) *
        K))
43  >              * Idf.log((numberOfDocuments - documentFrequency + 0.5d) /
        (documentFrequency + 0.5d));
44  >    }
45  >
46  >    // Second score function used in the second pass of retrieval
```

```
47 >    public double score(double tf, double docLength, Statement
   statement, Statement statement1, String term, String docno) {
48 >      ResultSet rs;
49 >      double pos_l, pos_r, dist_l = -1, dist_r = -1, sen_len, mid, ptf =
   0d;
50 >
51 >      try
52 >      {
53 >        rs = statement.executeQuery("select pos_l, pos_r, sen_length
   from index where doc='" + docno + "' and term='" + term + "'");
54 >
55 >        if (rs.next())
56 >        {
57 >          sen_len = rs.getDouble("sen_length");
58 >          mid = (sen_len - 1d) / 2d;
59 >
60 >          pos_l = rs.getDouble("pos_l");
61 >          if (rs.wasNull())
62 >          {
63 >            pos_l = -1;
64 >          }
65 >          else
66 >          {
67 >            dist_l = mid - pos_l;
68 >          }
69 >
70 >          pos_r = rs.getDouble("pos_r");
71 >          if (rs.wasNull())
72 >          {
73 >            pos_r = -1;
74 >          }
75 >          else
76 >          {
77 >            dist_r = mid - pos_r;
78 >          }
79 >
80 >          //ptf = Idf.log(1d + Weight(pos_l, pos_r, dist_l, dist_r,
   sen_len, mid));
81 >          ptf = Weight(pos_l, pos_r, dist_l, dist_r, sen_len, mid);
82 >        }
83 >        rs.close();
```

```
84 >      }
85 >      catch (SQLException e)
86 >      {
87 >        e.printStackTrace();
88 >      }
89 >
90 >      double K = k_1 * ((1 - b) + b * docLength / averageDocumentLength)
   + ptf * tf;
91 >      double bm25 = (ptf * tf * (k_3 + 1d) * keyFrequency / ((k_3 +
   keyFrequency) * K));
92 >      double idf = Idf.log((numberOfDocuments - documentFrequency +
   0.5d) / (documentFrequency + 0.5d));
93 >
94 >      //double ritf = Idf.log(1d + tf) / Idf.log(1d + docLength /
   numberOfUniqueTerms);
95 >      //double lrtf = tf * Idf.log(1d + averageDocumentLength /
   docLength);
96 >      //double qlf = 2d / (1d + Idf.log(1d + keyCount));
97 >      double qlf = 1d - Math.pow(0.5d / (0.5d + keyCount), 2d / 3d);
98 >
99 >      return ((1d - qlf) * bm25 + qlf) * idf;
100 >   }
101 >
102 >   double Weight(double pos_l, double pos_r, double dist_l, double
   dist_r, double sen_len, double mid)
103 >   {
104 >     double thresh = mid - sen_len / d + e;
105 >
106 >     return Idf.log(1d + sen_len) / Idf.log(11.5d) * (Kernel(dist_l,
   thresh) + Kernel(dist_r, thresh)) / (pos_l == -1 || pos_r == -1 ? 1d
   : 2d);
107 >   }
108 >
109 >   double Kernel(double dist, double thresh)
110 >   {
111 >     if (dist != -1)
112 >     {
113 >       if (dist >= thresh)
114 >       {
115 >         return 1d;
116 >       }
```

```
117 >       else
118 >       {
119 >         if (kernel == 0) return Gaussian(dist, thresh);
120 >         else if (kernel == 1) return Uniform(dist, thresh);
121 >         else if (kernel == 2) return Triangle(dist, thresh);
122 >         else if (kernel == 3) return Circle(dist, thresh);
123 >         else if (kernel == 4) return Cosine(dist, thresh);
124 >         else if (kernel == 5) return Quartic(dist, thresh);
125 >         else if (kernel == 6) return Epanechnikov(dist, thresh);
126 >         else if (kernel == 7) return Triweight(dist, thresh);
127 >       }
128 >     }
129 >
130 >     return 0d;
131 >   }
132 >
133 >   double Gaussian(double q, double m)
134 >   {
135 >     return 1d - Math.exp(Math.pow(q, 2) / (-2d * Math.pow(m, 2)));
136 >   }
137 >
138 >   double Uniform(double q, double m)
139 >   {
140 >     return 0d;
141 >   }
142 >
143 >   double Triangle(double q, double m)
144 >   {
145 >     return q / m;
146 >   }
147 >
148 >   double Circle(double q, double m)
149 >   {
150 >     return 1d - Math.sqrt(1d - Math.pow(q / m, 2));
151 >   }
152 >
153 >   double Cosine(double q, double m)
154 >   {
155 >     return 1d - (1d + Math.cos(q * Math.PI / m)) / 2d;
156 >   }
157 >
```

```
158 >   double Quartic(double q, double m)
159 >   {
160 >     return 1d - Math.pow(1d - Math.pow(q / m, 2), 2);
161 >   }
162 >
163 >   double Epanechnikov(double q, double m)
164 >   {
165 >     return Math.pow(q / m, 2);
166 >   }
167 >
168 >   double Triweight(double q, double m)
169 >   {
170 >     return 1d - Math.pow(1d - Math.pow(q / m, 2), 3);
171 >   }
172 >
173 >   /**
174 >    * Uses BM25 to compute a weight for a term in a document.
175 >    * @param tf The term frequency in the document
176 >    * @param docLength the document's length
177 >    * @param n_t The document frequency of the term
178 >    * @param F_t the term frequency in the collection
179 >    * @param keyFrequency the term frequency in the query
180 >    * @return the score assigned by the weighting model BM25.
181 >    */
182 >   public double score(
183 >           double tf,
184 >           double docLength,
185 >           double n_t,
186 >           double F_t,
187 >           double keyFrequency) {
188 >     double K = k_1 * ((1 - b) + b * docLength / averageDocumentLength)
    + tf;
189 >     return Idf.log((numberOfDocuments - n_t + 0.5d) / (n_t+ 0.5d)) *
190 >             ((k_1 + 1d) * tf / (K + tf)) *
191 >             ((k_3+1)*keyFrequency/(k_3+keyFrequency));
192 >   }
193 >
194 >   /**
195 >    * Sets the b parameter to BM25 ranking formula
196 >    * @param _b the b parameter value to use.
197 >    */
```

```
198 >    public void setParameter(double _b) {
199 >      this.b = _b;
200 >    }
201 >
202 40,109c232,237
203 <    /** The constant k_1.*/
204 <    private double k_1 = 1.2d;
205 <
206 <    /** The constant k_3.*/
207 <    private double k_3 = 8d;
208 <
209 <    /** The parameter b.*/
210 <    private double b;
211 <
212 <    /** A default constructor.*/
213 <    public BM25() {
214 <      super();
215 <      b=0.75d;
216 <    }
217 <    /**
218 <     * Returns the name of the model.
219 <     * @return the name of the model
220 <     */
221 <    public final String getInfo() {
222 <      return "BM25b"+b;
223 <    }
224 <    /**
225 <     * Uses BM25 to compute a weight for a term in a document.
226 <     * @param tf The term frequency in the document
227 <     * @param docLength the document's length
228 <     * @return the score assigned to a document with the given
229 <     *         tf and docLength, and other preset parameters
230 <     */
231 <    public double score(double tf, double docLength) {
232 <        double K = k_1 * ((1 - b) + b * docLength /
    averageDocumentLength) + tf;
233 <        return (tf * (k_3 + 1d) * keyFrequency / ((k_3 + keyFrequency) *
    K))
234 <                * Idf.log((numberOfDocuments - documentFrequency + 0.5d)
    / (documentFrequency + 0.5d));
235 <    }
```

```
236 <    /**
237 <     * Uses BM25 to compute a weight for a term in a document.
238 <     * @param tf The term frequency in the document
239 <     * @param docLength the document's length
240 <     * @param n_t The document frequency of the term
241 <     * @param F_t the term frequency in the collection
242 <     * @param keyFrequency the term frequency in the query
243 <     * @return the score assigned by the weighting model BM25.
244 <     */
245 <    public double score(
246 <      double tf,
247 <      double docLength,
248 <      double n_t,
249 <      double F_t,
250 <      double keyFrequency) {
251 <        double K = k_1 * ((1 - b) + b * docLength /
        averageDocumentLength) + tf;
252 <        return Idf.log((numberOfDocuments - n_t + 0.5d) / (n_t+ 0.5d)) *
253 <        ((k_1 + 1d) * tf / (K + tf)) *
254 <        ((k_3+1)*keyFrequency/(k_3+keyFrequency));
255 <    }
256 <
257 <    /**
258 <     * Sets the b parameter to BM25 ranking formula
259 <     * @param _b the b parameter value to use.
260 <     */
261 <    public void setParameter(double _b) {
262 <        this.b = _b;
263 <    }
264 <
265 <
266 <    /**
267 <     * Returns the b parameter to the BM25 ranking formula as set by
        setParameter()
268 <     */
269 <    public double getParameter() {
270 <        return this.b;
271 <    }
272 <
273 ---
274 >    /**
```

```
275 >     * Returns the b parameter to the BM25 ranking formula as set by
    setParameter()
276 >     */
277 >    public double getParameter() {
278 >      return this.b;
279 >    }
```

terrier-3.5/src/core/org/terrier/matching/models/DFI0.java

```
1  29a30,31
2  > import java.sql.Statement;
3  >
4  58a61,66
5  >    }
6  >
7  >    \@Override
8  >    public double score(double tf, double docLength, Statement
    statement, Statement statement1, String term, String docno) {
9  >      // TODO Auto-generated method stub
10 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/DFR_BM25.java

```
1  26a27,29
2  >
3  > import java.sql.Statement;
4  >
5  99a103,108
6  >    }
7  >
8  >    @Override
9  >    public double score(double tf, double docLength, Statement
    statement, Statement statement1, String term, String docno) {
10 >      // TODO Auto-generated method stub
11 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/DFRee.java

```
1  26a27,29
2  >
3  > import java.sql.Statement;
4  >
5  125a129,134
```

```
6  >    }
7  >
8  >    @Override
9  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
10 >      // TODO Auto-generated method stub
11 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/DFRWeightingModel.java

```
1  28c28
2  < import org.apache.log4j.Logger;
3  ---
4  > import java.sql.Statement;
5  29a30
6  > import org.apache.log4j.Logger;
7  284a286,291
8  >    }
9  >
10 >    @Override
11 >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
12 >      // TODO Auto-generated method stub
13 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/DirichletLM.java

```
1  26a27,28
2  > import java.sql.Statement;
3  >
4  65a68,73
5  >    }
6  >
7  >    @Override
8  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
9  >      // TODO Auto-generated method stub
10 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/DLH13.java

```
1  28a29,31
2  >
```

```
3  > import java.sql.Statement;
4  >
5  97a101,106
6  >   }
7  >
8  >   @Override
9  >   public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
10 >     // TODO Auto-generated method stub
11 >     return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/DLH.java

```
1  28a29,31
2  >
3  > import java.sql.Statement;
4  >
5  96a100,105
6  >   }
7  >
8  >   @Override
9  >   public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
10 >     // TODO Auto-generated method stub
11 >     return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/DPH.java

```
1  28a29,31
2  >
3  > import java.sql.Statement;
4  >
5  104a108,113
6  >
7  >   @Override
8  >   public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
9  >     // TODO Auto-generated method stub
10 >     return 0;
11 >   }
```

terrier-3.5/src/core/org/terrier/matching/models/Hiemstra_LM.java

```
26a27,28
> import java.sql.Statement;
>
113a116,121
>   }
>
>   @Override
>   public double score(double tf, double docLength, Statement
    statement, Statement statement1, String term, String docno) {
>     // TODO Auto-generated method stub
>     return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/IFB2.java

```
28a29,31
>
> import java.sql.Statement;
>
93a97,102
>   }
>
>   @Override
>   public double score(double tf, double docLength, Statement
    statement, Statement statement1, String term, String docno) {
>     // TODO Auto-generated method stub
>     return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/InB2.java

```
28a29,31
>
> import java.sql.Statement;
>
91a95,100
>   }
>
>   @Override
>   public double score(double tf, double docLength, Statement
    statement, Statement statement1, String term, String docno) {
>     // TODO Auto-generated method stub
>     return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/In_expB2.java

```
28a29,31
>
> import java.sql.Statement;
>
96a100,105
>   }
>
>   @Override
>   public double score(double tf, double docLength, Statement
    statement, Statement statement1, String term, String docno) {
>     // TODO Auto-generated method stub
>     return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/In_expC2.java

```
28a29,31
>
> import java.sql.Statement;
>
94a98,103
>   }
>
>   @Override
>   public double score(double tf, double docLength, Statement
    statement, Statement statement1, String term, String docno) {
>     // TODO Auto-generated method stub
>     return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/InL2.java

```
28a29,31
>
> import java.sql.Statement;
>
90a94,99
>   }
>
>   @Override
>   public double score(double tf, double docLength, Statement
    statement, Statement statement1, String term, String docno) {
>     // TODO Auto-generated method stub
```

```
11 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/Js_KLs.java

```
1  26a27,29
2  >
3  > import java.sql.Statement;
4  >
5  113a117,122
6  >    }
7  >
8  >    @Override
9  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
10 >      // TODO Auto-generated method stub
11 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/LemurTF_IDF.java

```
1  28a29,31
2  >
3  > import java.sql.Statement;
4  >
5  86a90,95
6  >    }
7  >
8  >    @Override
9  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
10 >      // TODO Auto-generated method stub
11 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/LGD.java

```
1  26a27,29
2  >
3  > import java.sql.Statement;
4  >
5  101a105,110
6  >    }
7  >
8  >    @Override
```

```
9  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
10 >      // TODO Auto-generated method stub
11 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/MDL2.java

```
1  28a29,30
2  > import java.sql.Statement;
3  >
4  159a162,167
5  >      return 0;
6  >    }
7  >
8  >    @Override
9  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
10 >      // TODO Auto-generated method stub
```

terrier-3.5/src/core/org/terrier/matching/models/ML2.java

```
1  28a29,30
2  > import java.sql.Statement;
3  >
4  151a154,159
5  >      return 0;
6  >    }
7  >
8  >    @Override
9  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
10 >      // TODO Auto-generated method stub
```

terrier-3.5/src/core/org/terrier/matching/models/PerFieldNormWeightingModel.java

```
1  27a28,29
2  > import java.sql.Statement;
3  >
4  181a184,188
5  >      return 0;
6  >    }
7  >    @Override
```

```
8  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
9  >      // TODO Auto-generated method stub
```

terrier-3.5/src/core/org/terrier/matching/models/PL2.java

```
1  29a30,31
2  > import java.sql.Statement;
3  >
4  103a106,110
5  >    }
6  >    @Override
7  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
8  >      // TODO Auto-generated method stub
9  >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/TF_IDF.java

```
1  28a29,31
2  >
3  > import java.sql.Statement;
4  >
5  117a121,125
6  >    }
7  >    @Override
8  >    public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
9  >      // TODO Auto-generated method stub
10 >      return 0;
```

terrier-3.5/src/core/org/terrier/matching/models/WeightingModel.java

```
1  29a30
2  > import java.sql.Statement;
3  48a50
4  >    protected int keyCount;
5  68a71,74
6  >
7  >    public double k_1, d, e, f;
8  >    public int kernel;
9  >
10 121a128,133
```

```
11 >   public double score(Posting p, Statement statement, Statement
       statement1, String term, String docno)
12 >   {
13 >     return this.score(p.getFrequency(), p.getDocumentLength(),
       statement,
14 >             statement1, term, docno);
15 >   }
16 >
17 159a172
18 >   public abstract double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno);
19 215c228,231
20 <
21 ---
22 >   public void setKeyCount(int count)
23 >   {
24 >     keyCount = count;
25 >   }
```

terrier-3.5/src/core/org/terrier/matching/models/XSqrA_M.java

```
1  26a27,29
2  >
3  > import java.sql.Statement;
4  >
5  108a112,117
6  >
7  >   @Override
8  >   public double score(double tf, double docLength, Statement
       statement, Statement statement1, String term, String docno) {
9  >     // TODO Auto-generated method stub
10 >     return 0;
11 >   }
```

terrier-3.5/src/core/org/terrier/matching/OldBasicMatching.java

```
1  28a29
2  > import java.sql.Statement;
3  32d32
4  <
5  260c260,261
6  <   public ResultSet match(String queryNumber, MatchingQueryTerms
       queryTerms) throws IOException {
```

```
---
>   public ResultSet match(String queryNumber, MatchingQueryTerms
    queryTerms, Statement statement, Statement statement1, double k_1,
    double d, double e, double f, double a,
>                          int kernel) throws IOException {
```

terrier-3.5/src/core/org/terrier/matching/PostingListManager.java

```
31a32
> import java.sql.Statement;
154c155
<   public PostingListManager(Index _index, CollectionStatistics _cs,
    MatchingQueryTerms mqt) throws IOException
---
>   public PostingListManager(Index _index, CollectionStatistics _cs,
    MatchingQueryTerms mqt, boolean log) throws IOException
170a172
>           mqt.length(),
179c181,186
<     logger.info("Query " + mqt.getQueryId() + " with "+
    mqt.getTerms().length +" terms has " + termPostings.size() + "
    posting lists");
---
>
>     if (log)
>     {
>       logger.info("Query " + mqt.getQueryId() + " with "+
    mqt.getTerms().length +" terms has " + termPostings.size() + "
    posting lists");
>     }
>
193c200
<   public void addSingleTerm(String queryTerm, double weight,
    EntryStatistics entryStats, WeightingModel[] wmodels) throws
    IOException
---
>   public void addSingleTerm(String queryTerm, double weight, int
    length, EntryStatistics entryStats, WeightingModel[] wmodels) throws
    IOException
216a224
>         w.setKeyCount(length);
352c360
```

```
25 <   public double score(int i)
26 ---
27 >   public double score(int i, double k_1)
28 358c366,367
29 <         for (WeightingModel w : termModels.get(i))
30 ---
31 >         for (WeightingModel w : termModels.get(i)) {
32 >           w.k_1 = k_1;
33 359a369,389
34 >         }
35 >         return score;
36 >       }
37 >
38 >     throw new IllegalArgumentException("Looking for posting list " + i
   + " out of " + (numTerms) + " posting lists.");
39 >   }
40 >
41 >   public double score(int i, Statement statement, Statement
   statement1, String docno, double k_1, double d, double e, double f,
   int kernel)
42 >   {
43 >     if (i >= 0)
44 >       if (i < numTerms)
45 >       {
46 >         double score = 0.0d;
47 >         for (WeightingModel w : termModels.get(i)) {
48 >           w.k_1 = k_1;
49 >           w.d = d;
50 >           w.e = e;
51 >           w.f = f;
52 >           w.kernel = kernel;
53 >           score += w.score(termPostings.get(i), statement, statement1,
   termStrings.get(i), docno);
54 >         }
```

terrier-3.5/src/core/org/terrier/matching/taat/Full.java

```
1 30a31,32
2 > import java.sql.Statement;
3 > import java.util.Arrays;
4 49a52
5 >
```

```
6  68c71,72
7  <    public ResultSet match(String queryNumber, MatchingQueryTerms
       queryTerms) throws IOException
8  ---
9  >    public ResultSet match(String queryNumber, MatchingQueryTerms
       queryTerms, Statement statement, Statement statement1, double k_1,
       double d, double e, double f, double a,
10 >                            int kernel) throws IOException
11 71a76
12 >      int[] docs;
13 73c78,79
14 <      plm = new PostingListManager(index, super.collectionStatistics,
       queryTerms);
15 ---
16 >      plm = new PostingListManager(index, super.collectionStatistics,
       queryTerms, true);
17 >
18 87c93
19 <        assignScores(i, (AccumulatorResultSet) resultSet,
       plm.getPosting(i));
20 ---
21 >        assignScores(i, (AccumulatorResultSet) resultSet,
       plm.getPosting(i), statement, statement1, null, k_1, d, e, f, a,
       kernel);
22 89c95,107
23 <
24 ---
25 >
26 >      ((AccumulatorResultSet)resultSet).Sort();
27 >      docs =
       Arrays.copyOfRange(((AccumulatorResultSet)resultSet).docids, 0, 1000);
28 >      Arrays.sort(docs);
29 >
30 >      plm = new PostingListManager(index, super.collectionStatistics,
       queryTerms, false);
31 >      plm.prepare(false);
32 >
33 >      for(int i=0; i< plm.size(); i++)
34 >      {
```

```
35 >       assignScores(i, (AccumulatorResultSet) resultSet,
       plm.getPosting(i), statement, statement1, docs, k_1, d, e, f, a,
       kernel);
36 >     }
37 >
38 98c116,117
39 <   protected void assignScores(int i, AccumulatorResultSet rs, final
       IterablePosting postings) throws IOException
40 ---
41 >   protected void assignScores(int i, AccumulatorResultSet rs, final
       IterablePosting postings, Statement statement, Statement statement1,
       int[] docs, double k_1, double d, double e, double f,
42 >                               double a, int kernel) throws IOException
43 106c125
44 <
45 ---
46 >
47 109d127
48 <       score = plm.score(i);
49 111c129,136
50 <       //logger.info("Docid=" + docid + " score=" + score);
51 ---
52 >       if (docs != null)
53 >         if (a != 0 && Arrays.binarySearch(docs, docid) >= 0)
54 >           score = a * plm.score(i, statement, statement1,
       index.getMetaIndex().getItem("docno", docid), k_1, d, e, f, kernel);
55 >         else
56 >           score = 0;
57 >       else
58 >         score = (1 - a) * plm.score(i, k_1);
59 >
```

terrier-3.5/src/core/org/terrier/matching/taat/FullNoPLM.java

```
1 30a31
2 > import java.sql.Statement;
3 35d35
4 <
5 74c74,75
6 <   public ResultSet match(String queryNumber, MatchingQueryTerms
       queryTerms) throws IOException
7 ---
```

```
8 >    public ResultSet match(String queryNumber, MatchingQueryTerms
      queryTerms, Statement statement, Statement statement1, double k_1,
      double d, double e, double f, double a,
9 >                           int kernel) throws IOException
```

terrier-3.5/src/core/org/terrier/matching/TRECResultsMatching.java

```
1 31a32
2 > import java.sql.Statement;
3 222c223
4 <    public ResultSet match(String _qid, MatchingQueryTerms mqt) throws
      IOException {
5 ---
6 >    public ResultSet match(String _qid, MatchingQueryTerms mqt,
      Statement statement, Statement statement1, double k_1, double d,
      double e, double f, double a, int kernel) throws IOException {
```

terrier-3.5/src/core/org/terrier/matching/tsms/RequiredTermModifier.java

```
1 27a28
2 > import java.sql.Statement;
3 125a127,132
4 >      // TODO Auto-generated method stub
5 >      return 0;
6 >    }
7 >
8 >    @Override
9 >    public double score(double tf, double docLength, Statement
      statement, Statement statement1, String term, String docno) {
```

terrier-3.5/src/core/org/terrier/matching/tsms/TermInFieldModifier.java

```
1 26a27,29
2 >
3 > import java.sql.Statement;
4 >
5 206a210,215
6 >    }
7 >
8 >    @Override
9 >    public double score(double tf, double docLength, Statement
      statement, Statement statement1, String term, String docno) {
10 >      // TODO Auto-generated method stub
```

```
11 >     return 0;
```

terrier-3.5/src/core/org/terrier/querying/Decorate.java

```
1  32a33
2  > import java.sql.Statement;
```

terrier-3.5/src/core/org/terrier/querying/Manager.java

```
1  30a31
2  > import java.sql.Statement;
3  40d40
4  <
5  617c617
6  <   public void runMatching(SearchRequest srq)
7  ---
8  >   public void runMatching(SearchRequest srq, Statement statement,
       Statement statement1, double k_1, double d, double e, double f,
       double a, int kernel)
9  624d623
10 <
11 676c675
12 <         ResultSet outRs = matching.match(rq.getQueryID(), mqt);
13 ---
14 >         ResultSet outRs = matching.match(rq.getQueryID(), mqt,
       statement, statement1, k_1, d, e, f, a, kernel);
```

terrier-3.5/src/core/org/terrier/querying/Process.java

```
1  27a28,30
2  >
3  > import java.sql.Statement;
4  >
```

terrier-3.5/src/core/org/terrier/querying/QueryExpansion.java

```
1  30a31
2  > import java.sql.Statement;
3  35d35
4  <
5  134c134
6  <     double totalDocumentLength = 0;
7  ---
```

```
>     double totalDocumentLength = 0;
329c329
<     manager.runMatching(q);
---
>     manager.runMatching(q, null, null, 1.2, 0, 0, 0, 0, 0);
```

terrier-3.5/src/core/org/terrier/structures/indexing/DocumentPostingList.java

```
33a34,37
> import java.io.PrintWriter;
> import java.sql.SQLException;
> import java.sql.Statement;
> import java.sql.ResultSet;
102a107,121
>   }
>
>   public void insert(final String term, final Statement statement,
    final int pos, final int length)
>   {
>     occurrences.adjustOrPutValue(term,1,1);
>     documentLength++;
>
>     /*try
>     {
>       statement.execute("insert into temp values('" + term + "'," +
    Math.abs(((length - 1d) / 2 - pos)) + "," + length + ")");
>     }
>     catch (SQLException e)
>     {
>       System.out.println(e.getMessage());
>     }*/
```

terrier-3.5/src/core/org/terrier/structures/TRECQuery.java

```
127,138c127,140
<
<             if (queryTokenizer.inDocnoTag()) {
<               //The tokenizer is constructed from the trimmed version
    of the contents
<               //of the query number tag, so that the last token
    extracted from it, is
<               //always the query number, and not an empty string
<               StringTokenizer docnoTokens =
```

```
8  <                   new StringTokenizer(token.trim(), " ");
9  <                 while (docnoTokens.hasMoreTokens())
10 <                   docnoToken = docnoTokens.nextToken().trim();
11 <               } else if (queryTokenizer.inTagToProcess()) {
12 <                 // Removed the code that checks if "description" and
13 <                 // "narrative" appear in "desc" and "narr", respective.
14 ---
15 >               if (queryTokenizer.inDocnoTag()) {
16 >                 //The tokenizer is constructed from the trimmed version
       of the contents
17 >                 //of the query number tag, ignoring the token Number:
18 >                 StringTokenizer docnoTokens =
19 >                   new StringTokenizer(token.trim(), " ");
20 >                 while (docnoTokens.hasMoreTokens())
21 >                 {
22 >                   String tok = docnoTokens.nextToken().trim();
23 >                   if (! tok.equalsIgnoreCase("number"))
24 >                     docnoToken = tok;
25 >                 }
26 >               } else if (queryTokenizer.inTagToProcess()) {
27 >                 // Removed the code that checks if "description" and
28 >                 // "narrative" appear in "desc" and "narr", respective.
29 144,146c146,147
30 <                   .toUpperCase()
31 <                   .equals("DESC")
32 <                   && token.toUpperCase().equals("DESCRIPTION"))
33 ---
34 >                   .equalsIgnoreCase("DESC")
35 >                   && token.equalsIgnoreCase("DESCRIPTION"))
36 150,152c151,152
37 <                   .toUpperCase()
38 <                   .equals("NARR")
39 <                   && token.toUpperCase().equals("NARRATIVE"))
40 ---
41 >                   .equalsIgnoreCase("NARR")
42 >                   && token.equalsIgnoreCase("NARRATIVE"))
43 163c163,165
44 <             vecStringIds.add(docnoToken.trim());
45 ---
46 >             if (docnoToken == null)
47 >               throw new IOException("No id tag found for this query");
```

```
48 >            vecStringIds.add(docnoToken);
```

terrier-3.5/src/core/org/terrier/terms/BaseTermPipelineAccessor.java

```
1  27a28,29
2  > import java.sql.Statement;
3  >
4  88c90
5  <   public void processTerm(String t)
6  ---
7  >   public void processTerm(String t, Statement statement, int pos, int
       length)
8  103c105
9  <     pipeline_first.processTerm(t);
10 ---
11 >     pipeline_first.processTerm(t, null, 0, 0);
```

terrier-3.5/src/core/org/terrier/terms/CropTerm.java

```
1  27a28,30
2  >
3  > import java.sql.Statement;
4  >
5  56c59
6  <   public void processTerm(String t)
7  ---
8  >   public void processTerm(String t, Statement statement, int pos, int
       length)
9  62c65
10 <     next.processTerm(t);
11 ---
12 >     next.processTerm(t, statement, pos, length);
```

terrier-3.5/src/core/org/terrier/terms/DumpTerm.java

```
1  28a29,30
2  > import java.sql.Statement;
3  >
4  45c47
5  <   public void processTerm(String t)
6  ---
7  >   public void processTerm(String t, Statement statement, int pos, int
       length)
```

```
8  50c52
9  <     next.processTerm(t);
10 ---
11 >     next.processTerm(t, statement, pos, length);
```

terrier-3.5/src/core/org/terrier/terms/NoOp.java

```
1  28a29,30
2  > import java.sql.Statement;
3  >
4  54c56
5  <   public final void processTerm(final String t)
6  ---
7  >   public final void processTerm(final String t, final Statement
       statement, final int pos, final int length)
8  58c60
9  <         next.processTerm(t);
10 ---
11 >         next.processTerm(t, statement, pos, length);
```

terrier-3.5/src/core/org/terrier/terms/SkipTermPipeline.java

```
1  29a30,31
2  > import java.sql.Statement;
3  >
4  30a33
5  >
6  89c92
7  <   public void processTerm(String term) {
8  ---
9  >   public void processTerm(String term, Statement statement, int pos,
       int length) {
10 94c97
11 <       last.processTerm(term);
12 ---
13 >       last.processTerm(term, statement, pos, length);
14 99c102
15 <       next.processTerm(term);
16 ---
17 >       next.processTerm(term, statement, pos, length);
```

terrier-3.5/src/core/org/terrier/terms/StemmerTermPipeline.java

```
29a30,31
> import java.sql.Statement;
>
55c57
<   public void processTerm(String t)
---
>   public void processTerm(String t, Statement statement, int pos, int
    length)
59c61
<     next.processTerm(stem(t));
---
>     next.processTerm(stem(t), statement, pos, length);
```

terrier-3.5/src/core/org/terrier/terms/Stopwords.java

```
29a32
> import java.sql.Statement;
172c175
<   public void processTerm(final String t)
---
>   public void processTerm(final String t, final Statement statement,
    final int pos, final int length)
176c179
<     next.processTerm(t);
---
>     next.processTerm(t, statement, pos, length);
```

terrier-3.5/src/core/org/terrier/terms/TermPipeline.java

```
26a27,28
>
> import java.sql.Statement;
41c43
<   void processTerm(String t);
---
>   void processTerm(String t, Statement statement, int pos, int length);
```

Appendix E

Terrier Properties Files

In order for Terrier to index and run retrieval, it needs a properties file. This file is located in the terrier-3.5/etc directory and is named terrier.properties. The properties file determines Terrier run-time properties such as which tags in a document to process and ignore. When working with a collection, Terrier needs the specific properties for that collection in order to perform optimally. The properties used in these experiments for each of the collections are as follows:

WT2g, WT10g, Gov2

```
1  #default controls for query expansion
2  querying.postprocesses.order=QueryExpansion
3  querying.postprocesses.controls=qe:QueryExpansion
4
5  #default and allowed controls
6  querying.default.controls=
7  querying.allowed.controls=qe,start,end,qemodel
8
9  #document tags specification
10 #for processing the contents of
11 #the documents, ignoring DOCHDR
12 TrecDocTags.doctag=DOC
13 TrecDocTags.idtag=DOCNO
14 TrecDocTags.skip=DOCHDR
15
16 #query tags specification
17 TrecQueryTags.doctag=TOP
18 TrecQueryTags.idtag=NUM
```

```
TrecQueryTags.process=TOP,NUM,TITLE
TrecQueryTags.skip=DESC,NARR

#stop-words file
stopwords.filename=stopword-list.txt

#the processing stages a term goes through
termpipelines=Stopwords,PorterStemmer
```

disk4+5

```
#default controls for query expansion
querying.postprocesses.order=QueryExpansion
querying.postprocesses.controls=qe:QueryExpansion

#default and allowed controls
querying.default.controls=
querying.allowed.controls=qe,start,end,qemodel

#document tags specification
#for processing the contents of
#the documents, ignoring DOCHDR
TrecDocTags.doctag=DOC
TrecDocTags.idtag=DOCNO
TrecDocTags.skip=DOCHDR
TrecDocTags.process=TEXT,H3,DOCTITLE,HEADLINE,TTL

#query tags specification
TrecQueryTags.doctag=TOP
TrecQueryTags.idtag=NUM
TrecQueryTags.process=TOP,NUM,TITLE
TrecQueryTags.skip=DESC,NARR

#stop-words file
stopwords.filename=stopword-list.txt

#the processing stages a term goes through
termpipelines=Stopwords,PorterStemmer
```

Blogs06

```
#default controls for query expansion
querying.postprocesses.order=QueryExpansion
```

```
querying.postprocesses.controls=qe:QueryExpansion

#default and allowed controls
querying.default.controls=
querying.allowed.controls=qe,start,end,qemodel

#document tags specification
#for processing the contents of
#the documents, ignoring DOCHDR
TrecDocTags.doctag=DOC
TrecDocTags.idtag=DOCNO
TrecDocTags.skip=DOCHDR,FEEDNO,FEEDURL,BLOGHPNO,BLOGHPURL,PERMALINK,DATE_XML

indexing.singlepass.max.postings.memory=500000000
indexer.meta.forward.keys=docno
indexer.meta.forward.keylens=31
indexer.meta.reverse.keys=docno

#query tags specification
TrecQueryTags.doctag=TOP
TrecQueryTags.idtag=NUM
TrecQueryTags.process=TOP,NUM,TITLE
TrecQueryTags.skip=DESC,NARR

#stop-words file
stopwords.filename=stopword-list.txt

#the processing stages a term goes through
termpipelines=Stopwords,PorterStemmer
```

Appendix F

Generating the Primary Index

1. Copy the properties specified in Appendix E into the terrier.properties file in the terrier-3.5/etc directory.
2. Place the collection to index in the terrier-3.5/collections directory.
3. Enter "./trec_terrier.sh -i" in command line in the terrier-3.5/bin directory.

Appendix G

Generating the Secondary Index

1. Setup the Query Term Placement Parser by running through the steps 1 to 15 in Appendix B.1.

2. Enter "java -cp .:stanford-postagger-3.3.1.jar:h2-1.3.174.jar RelParser [Data] -1 -i" into the command line. The [Data] parameter is mandatory. [Data] is the name of the collection directory, ex. wt2g. For example, to generate the secondary index for the WT10g collection, enter "java -cp .:stanford-postagger-3.3.1.jar:h2-1.3.174.jar RelParser wt10g -1 -i" into the command line.

3. Copy the index.h2.db generated by the Query Term Placement Parser into the terrier-3.5/var directory.

Appendix H

Running Retrieval on Terrier

1. Modify Terrier 3.5 as specified in Appendix D.

2. Copy the properties specified in Appendix E into the terrier.properties file in the terrier-3.5/etc directory.

3. Generate the primary and secondary indexes as specified in Appendix F and Appendix G.

4. Take all of the topics of the collection as defined in Section 5.1 and concatenate them into a single file named according to the collection. For example, topics.wt10g should contain the topics 451-550 for the WT10g collection. Place this file in the terrier-3.5/collections directory.

5. Enter "bin/trec_terrier.sh -r -Dtrec.model=BM25 -Dtrec.topics=collections/topics.[Data] -k1 1.2 -dd 2 -ee 3 -aa 0.2 -kk [Kernel] -fold 0 -c [c]" into the command line. The [Data], [Kernel], and [c] parameters are mandatory. [Data] is the name of the collection directory, ex. wt2g. [Kernel] is the kernel function to use for the retrieval. The options for [Kernel] are as follows: Baseline (Unmodified Terrier) = -1, Gaussian = 0, Uniform = 1, Triangle = 2, Circle = 3, Cosine = 4, Quartic = 5, Epanechikov = 6, Triweight = 7. [c] is a tuning parameter for BM25. The values of [c] used in these experiments are as follows, WT2g = 0.2, WT10g = 0.3, disk4+5 = 0.3, Blogs06 = 0.2, Gov2 = 0.4. For example, to run retrieval on WT10g using

the Gaussian kernel function, enter "bin/trec_terrier.sh -r -Dtrec.model=BM25 -Dtrec.topics=collections/topics.wt10g -k1 1.2 -dd 2 -ee 3 -aa 0.2 -kk 0 -fold 0 -c 0.3" into the command line.