Extracting Significant Phrases from Documents in English and Chinese

Yuan Jenq (Oliver) Lui
St Hugh's College

Thesis submitted for the degree of Doctor of Philosophy
Michaelmas Term 2008

In memory of my father (1949-2006)

Acknowledgments

I would like to thank my family (especially my father) and friends for their support, Prof Richard Brent and Dr Ani Calinescu for their supervision, Prof Stephen Pulman and Prof Beniamino Di Martino for examining my work, Dr Stephen Clark and Dr Vasile Palade for their comments on my work, Prof Laurence Yang for his help, and Dr Jeff Sanders for checking the Z specification in the Appendix.

Declaration

The following papers have been published during my doctoral studies at the University of Oxford:

Journals

- Yuan Lui, Ani Calinescu, Richard Brent and Laurence Yang, Extraction of Significant Phrases from Documents in English and Chinese, Cybernetics and Systems, 2008 (submitted)

- Yuan Lui, Extraction of Significant Phrases from Text, International Journal of Computer Science, Vol. 2, No. 2, pp. 101-109, 2007

Conferences

- Yuan Lui, Richard Brent and Ani Calinescu, Extracting Significant Phrases from Text, Proceedings of IEEE Data Mining and Information Retrieval (DMIR-07), Ontario, Canada, IEEE Computer, pp. 361-366, 2007

- Yuan Lui, An Improved Keyphrase Extraction Algorithm, Proceedings of PREP2005, Lancaster, UK, 2005

Research Report

- Yuan Lui, Learning to Extract Significant Phrases from Text, Research Report RR-07-01, Oxford University Computing Laboratory, UK, 2007

Abstract

Prospective readers can quickly determine whether a document is relevant to their information need if the significant phrases (or keyphrases) in this document are provided. Although keyphrases are useful, not many documents have keyphrases assigned to them, and manually assigning keyphrases to existing documents is costly. Therefore, there is a need for automatic keyphrase extraction.

This thesis proposes a new domain-independent keyphrase extraction algorithm. The algorithm approaches the problem of keyphrase extraction as a classification task, and uses a combination of statistical and text processing techniques, a new set of attributes, and a new machine learning method to distinguish keyphrases from non-keyphrases. The experiments indicate that this algorithm performs better than other keyphrase extraction tools and that it significantly outperforms Microsoft Word 2000's AutoSummarize feature. The domain independence of this algorithm has also been validated on a set of heterogeneous documents. To explore the use of this algorithm in a language other than English, we make minimal changes to this algorithm and apply it to Chinese documents. This variation of the algorithm has been tested on a set of Chinese documents on different subject areas, and the experiments show that it can extract keyphrases from these documents. This confirms the domain independence of this algorithm, and suggests that the keyphrase extraction algorithm proposed in this thesis can be applied to another language. This is, as far as we know, the first time a keyphrase extraction algorithm has been validated and evaluated on different domains and different languages.

Contents

Figures

Tables

CHAPTER ONE
Introduction

1.1 Background

With the proliferation of the Internet and the huge numbers of *documents*[1] it contains, the provision of condensed versions (or summaries) of these documents has become more and more important. Prospective readers can quickly determine whether a document is relevant to their information need if the significant phrases (or *keyphrases*) in this document are provided. Keyphrases give a short summary of the document and provide supplementary information for the readers, in addition to titles and abstracts. Even though keyphrases are useful, only a small minority of documents have keyphrases assigned to them, and manually assigning keyphrases to existing documents is rather costly. Therefore, there is a need for automatic keyphrase extraction [34, 66-69, 105-108].

Automatic keyphrase extraction (*keyphrase extraction* for short) is the identification of the most important phrases within the body of a document by computers rather than humans. It normally involves the use of statistical information. There is no controlled vocabulary list, so in theory any phrase within the body of the document can be identified as a keyphrase. When authors assign keyphrases without a controlled vocabulary list, typically 70-90% of their keyphrases appear somewhere in their documents [105]. Keyphrases are similar to keywords, except that the document is summarized by a set of phrases rather than words.

[1] "Document" is regarded as being synonymous with "text" in this thesis. We ignore non-text elements, e.g. graphics, sound and video, though a document might contain them in its body.

Keyphrase extraction involves the use of computers to extract keyphrases from documents. However, it is very difficult to get computers to understand the meaning (or semantics) of human languages, so statistical methods have been used instead. Luhn [64] used statistical methods to automatically extract significant words from documents and it was a big success. Since then, statistical methods have been increasingly used by the information retrieval (IR) community, and they have proved effective in various contexts [4, 12, 60, 89].

Keyphrase extraction is a classification task: a document could be seen as a set of phrases, and a keyphrase extraction algorithm should correctly classify a phrase as a keyphrase or a non-keyphrase. Machine learning techniques can automate this task if they are provided with a set of training data composed of both keyphrase examples and non-keyphrase examples. The data are used to train the algorithm to distinguish keyphrases from non-keyphrases. The resulting algorithm can then be applied to new (i.e. previously unseen) documents for keyphrase extraction. Previous work shows that the training data and the new documents need not be from the same domain, though the performance of the algorithm can be boosted significantly if they are [34].

A number of keyphrase extraction algorithms have been proposed [17, 27, 34, 71, 93, 107, 118, 123]. Some of these algorithms use domain-dependent information to extract keyphrases and are tied to a specific domain [71, 118]; some use purely statistical techniques [34, 93, 107]; some use a different corpus to evaluate their performance [17, 27, 123], so it is very difficult to directly compare their results. In addition, all these algorithms are intended for English documents and have only been tested on these documents. For details of these keyphrase extraction algorithms, see Section 2.2.

We would like to develop a domain-independent keyphrase extraction algorithm, to use statistical and text processing techniques to extract keyphrases from heterogeneous documents, to validate our algorithm by using the same training and testing data as in some existing algorithms so that our performance can be directly compared with theirs, and to explore the use of our algorithm in a language other than English. For details of the corpora used in our experiments, see Section 4.2, Section 4.4.7 and Section 5.5.

1.2 Applications of Keyphrases

Keyphrases have a wide range of applications: summarization, indexing, compression, author assistance, query reformulation, and classification and clustering [39, 52, 108].

- Keyphrases provide a short summary of the document. This could be a useful feature of web browsers. A browser could provide an option to allow users to summarize the current page, or provide the site map of a given web site by traversing the site and summarizing each page in a few keyphrases.

- A document could be summarized by a number of keyphrases. These phrases could be used as the index terms to represent the content of that document. Jones and Staveley [52] suggest that keyphrase-based indexing can be as effective as full text indexing, although their system has only been tested on a small scale. Similarly, a back-of-the-book index could be automatically generated by extracting keyphrases from each section or chapter and then merging all the keyphrases together to form the index of that book.

- The summarizing feature of keyphrases means that fewer words can be used to represent the content of a document.

- As we will see later, the quality of machine-extracted keyphrases is not as good as that of author-assigned keyphrases. Nevertheless, machine-extracted keyphrases can provide valuable information about the content of a document, and give the author a useful starting point for further manual refinement when author-assigned keyphrases are not available.

- Keyphrases could also be used for query reformulation. Web users may sometimes provide the search engine with queries that are too general. This often results in a long list of matching documents. The search engine could help the users to narrow down the list by providing a keyphrase list for each returned document so that the users can reformulate their query by adding new words to it. This is similar to the idea of relevance feedback in IR [87].

- Keyphrases could be used for document classification and clustering. Classification involves the assignment of documents to one of the predefined categories, while clustering involves the division of documents into groups such that the similarity between groups (inter-group) is minimized and the similarity within groups (intra-group) is maximized.

1.3 Definitions

This section defines some of the important terms which will be used in this thesis. Terms that are used only in one chapter are defined in that chapter.

1.3.1 Stopwords

Stopword lists contain words with high frequency and little semantic value. These words are generally not used for searching. Articles, prepositions and conjunctions are typical examples of stopwords. Different stopword lists have been proposed [32, 110]. The British National Corpus [11] may be consulted about the frequency of common English words for potential stopwords. Stopword lists will be discussed again in Section 2.3.

1.3.2 Index Terms

Index terms (*terms* for short) or keywords are used to represent (or describe) the content of a document or query. Not all the words in a document or query text are equally important. Some words contain more meaning than others. Nouns (or groups of nouns) are usually the ones which are the most representative of text content [2]. Less important words could be removed from the text for efficiency purposes. This usually involves three activities: elimination of stopwords, stemming, and detection of equivalent stems [89, 95, 110].

1. A stopword list is used to eliminate high frequency words such as 'a', 'is', and 'the'.

2. A stemming algorithm is then applied to the remaining words to reduce variants of a word to a single form. For example, 'connect' is the stem of the forms 'connect', 'connected', 'connecting', 'connection' and 'connections'.

3. Multiple occurrences of a given stem (i.e. words with the same stem) are combined into a single term for content representation. In other words, 'connected', 'connecting', 'connection' and 'connections' are combined into the term 'connect'.

As a result, terms are usually stems rather than full words [85]. Suppose a document contains the sentence 'People in need of information require effective retrieval services' [89]. Words such as 'in', 'need', 'of', and 'require' might be eliminated through a stopword list. The remaining words are then stemmed and combined into terms representing this sentence: 'People', 'inform', 'effect', 'retriev', and 'service' (which correspond to the words: 'People', 'information', 'effective', 'retrieval', and 'services'). Index terms will be discussed again in Section 2.3.

1.3.3 Phrases and Term Phrases

Phrases consist of one or more words, e.g. ‘effective retrieval services’.

Term phrases consist of one or more terms, e.g. ‘effect retriev service’ (which corresponds to the phrase: ‘effective retrieval services’).

1.3.4 Inverted Files

An *inverted file* (or inverted index) is a mechanism for indexing the document collection to speed up the searching process. This file consists of two elements: term and document reference number (i.e. the documents in which the term occurs) [90]. Each term is used as a key to access the corresponding documents. Table 1.1 shows an example of the inverted file. However, inverted files have a big disadvantage: the addition of new documents to the collection can be costly because not only must a new document be placed in the collection, but the index entries relating to this document must also be updated [90, 115].

Table 1.1: Inverted file

Term	Document Reference Number
Algorithm	3
Computer	1, 3, 5
Network	2, 4
Programming	1, 6

1.3.5 TF×IDF

The attribute *TF×IDF* [2, 85, 88, 95] consists of two parts. The first part is *term frequency* (TF) which is the frequency of a term in the document. The more often a term occurs in the document, the more likely it is to be important for that document. The second part is *inverse document frequency* (IDF), or collection frequency, which is the rarity of a term across the collection (or corpus). A term that occurs in only a few documents is often more valuable than a term that occurs in many documents. The *standard TF* and the *standard IDF* of a term *T* in a document *D* is calculated by:

$$\text{Standard TF} = \text{no. of occurrences of } T \text{ in } D \tag{1.1}$$

$$\text{Standard IDF} = \log \frac{\text{no. of documents in collection}}{\text{no. of documents } T \text{ occurs in}} \tag{1.2}$$

Despite the popularity of TF and IDF, they do not have a universal definition. Different definitions of TF and IDF have been proposed [88]. In addition to the standard TF and standard IDF, the normalized TF and Kea's IDF will be discussed in this thesis. Please refer to Section 2.2.2 for further information about Kea's IDF, and to Section 2.3.3 for the normalized TF.

1.3.6 Position

The attribute *position* [34, 71, 107] is the position where a term first appears in the document. A term that occurs at the beginning of the document is often more valuable than a term that occurs at the end of that document. Turney [107] defines *position* as the actual position where a term *T* first appears in a document *D* (for details, see Section 2.2.1), but we think it is more reasonable to normalize the actual position by the document length:

$$\text{Position} = \frac{\text{position where } T \text{ first appears in } D}{\text{length of } D} \tag{1.3}$$

Number of words has been used as the unit of measurement for *position*.

1.3.7 Title

The attribute *title* is a flag that indicates if a term appears in the title of the document. This assumes that the author of a document reveals the most important concepts in the title of his work [112]. A term that occurs in the title of the document is often more valuable than a term that does not. Titles may not provide enough information on their own, but they may contain some important words. In fact, it has been reported that the use of abstracts in addition to titles brings substantial advantages in retrieval effectiveness and that the additional utilization of the full texts of the documents appears to produce little improvement over titles and abstracts alone in most subject areas [90]. If a term is found in the title, *title* is set to 1; otherwise, it is set to 0.

1.3.8 Proper Noun

The attribute *proper noun* is a flag that indicates if a term is a proper noun. Proper nouns could sometimes be valuable. They might not be valuable in domains such as academic papers which tend to contain many proper nouns, especially in the References section. However, they might be valuable in domains such as news where they tend to occur less frequently. If a term is a proper noun, *proper noun* is set to 1; otherwise, it is set to 0.

1.3.9 Number of Terms

The attribute *number of terms* is the number of terms in a term phrase. A term tends to occur more frequently in the document than a two-word term phrase, and a two-word term phrase tends to occur more frequently than a three-word term phrase. If a three-word term phrase occurs the same number of times as a two-word term phrase, it is likely to be more important. The *number of terms* of a term phrase *P* is given by:

$$\text{Number of Terms} = \text{no. of terms in } P \tag{1.4}$$

1.3.10 Document Length

The attribute *document length* [85] is the length of a document. The *document length* of a document *D* is calculated by:

$$\text{Document length} = \text{no. of words in } D \tag{1.5}$$

1.3.11 Indicator Phrases

Paice [76] suggests the use of *indicator phrases* for summarizing documents. Indicator phrases contain words that are likely to accompany indicative or informative summary material, e.g. 'The objective of this study is…', 'The findings from our research show…', and 'We may conclude that…', which may help to identify sentences about the topic, aim or findings of a document [49].

1.3.12 Recall and Precision

Recall and *precision* are often used as measures of retrieval *effectiveness*. Effectiveness refers to the ability of an IR system to satisfy the user in terms of the relevance of documents retrieved [110]. Recall is the proportion of the relevant documents that are retrieved in a search, while precision is the proportion of the documents retrieved in a search that are relevant. The standard recall and the standard precision are given by [88, 95]:

$$\text{Recall} = \frac{\text{no. of relevant documents retrieved}}{\text{no. of relevant documents in collection}} \tag{1.6}$$

$$\text{Precision} = \frac{\text{no. of relevant documents retrieved}}{\text{no. of documents retrieved}} \tag{1.7}$$

Recall and precision will be discussed again in Section 2.3.5 and Section 4.3.3.

1.4 Outline

This section provides an outline of our research work and this thesis.

1.4.1 Research

This thesis discusses a new domain-independent keyphrase extraction algorithm called *KE*. KE is not tied to a specific domain; it is designed to summarize a given document, which is written in English and can be on any topic[2], in a few keyphrases automatically extracted from the body of that document.

The selection of relevant attributes is probably the most important factor in determining the effectiveness of a keyphrase extraction algorithm. Many attributes have been evaluated in our experiments, e.g. the length of a document, the number of characters in a term, the number of occurrences of a term throughout the collection, etc. However, only five of them have been found useful for keyphrase extraction: *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*.

The KE algorithm consists of seven steps which could be grouped into three activities (see Section 3.3 for a detailed description of KE):

[2] Excluding poetry and other similar works of literature.

1. KE selects words and phrases from the input document, and uses a vector of terms and a vector of term phrases to represent this document. Each term is characterized by four attributes: *TF×IDF*, *position*, *title*, and *proper noun*. A score is assigned to each term based on these attributes. The weights associated with these attributes are tuned by a machine learning method using keyphrase and non-keyphrase examples during training. After training, they are frozen and used to make predictions. For details of the training of KE, see Section 3.4. The term phrase vector is similar to the term vector, except that each term phrase is characterized by a different set of attributes: *TF×IDF*, *position*, *title* and *number of terms*.

2. A one-to-one relationship is established between the terms and the term phrases. For each term, KE finds all the term phrases that contain the term, and link it with the highest scoring term phrase. More than one term may link to the same term phrase. If that is the case, the term phrase will be linked to the highest scoring term. The result is a list of term phrases ordered by the scores of the corresponding terms. This is because it is generally preferable to represent documents and measure the importance of each representation element using single terms rather than term phrases [88].

3. The list of term phrases is then used to generate the output keyphrases. For each term phrase in the list, KE finds the most frequent corresponding phrase in the document. The result is a list of phrases. If a phrase occurs within another phrase, it will only be accepted as a keyphrase if it is ranked higher; otherwise, it will be deleted from the output list.

Table 1.2: Differences between KE, GenEx and Kea

	KE	GenEx	Kea
Techniques	Statistical and text processing; uses part-of-speech tagging to select candidate phrases	Statistical	Statistical
Attributes	*TF×IDF*, *position*, *title*, *proper noun* and *number of terms*	Many more attributes; does not use *TF×IDF* and *title*	TF×IDF and distance (same as *position*)
Learning method	Neural network[3]	Genetic algorithm	Naïve Bayes
Algorithm	Seven steps; takes both words and phrases as candidate phrases	More complicated; ten steps, considers both words and phrases, and involves many post-processing tasks	Simple; selects only phrases as candidate phrases, so does not involve any linking between words and phrases

KE is based on two previously published keyphrase extraction algorithms, *GenEx* [106, 107] and *Kea* [34], but differs from them in several ways. Please refer to Section 2.2.1 for further information about the GenEx algorithm, to Section 2.2.2 for the Kea algorithm, and to Section 3.5 for a detailed comparison between KE, GenEx and Kea. Table 1.2 summarizes the differences between KE, GenEx and Kea. The experimental results summarized in Table 4.17 (see Section 4.4.5) suggest that these differences make KE a better algorithm than either GexEx or Kea.

[3] "Neural network" is regarded as being synonymous with "artificial neural network" in this thesis.

We need some way to evaluate the performance of KE. KE has been tested on two different corpora. The first corpus is the same as the one used in GenEx and Kea, and it has been used to train and test KE in all our experiments (except the one discussed in Section 4.4.7). The criteria used for evaluating the output keyphrases are also the same as in GenEx and Kea, so direct comparison is possible. For details of this corpus and the evaluation method used, see Section 4.2 and Section 4.3. The second corpus is different and larger than the first one, and it has been used to test the generalization performance of KE. The evaluation criteria used for this corpus are the same as for the first corpus. For details of this corpus, see Section 4.4.7.

Testing has been carried out to validate and evaluate the KE algorithm. We have evaluated the individual performance of different attributes and the performance of different combinations of attributes. The experiments suggest that *position* gives the best individual performance and that the best combination of attributes involves *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. In addition, we have compared different combinations of *TF×IDF*, and found that the standard TF and Kea's IDF gives the best performance. The experiments also indicate that KE performs better than other keyphrase extraction tools, including GenEx and Kea, and that it significantly outperforms Microsoft Word 2000's AutoSummarize feature. We have tried using the C4.5 decision tree learning method to tune KE, but the experiments show that neural networks are better for keyphrase extraction than this method. The domain independence of KE has also been confirmed in our experiments using the second corpus. For further information about the experimental results relevant to KE, see Section 4.4.

To extend the use of the KE algorithm to another language, we make minimal changes to KE and apply it to Chinese documents. For details of these changes, see Section 5.4. For convenience, we use *KEC* to refer to this variation of KE for Chinese documents. Because of a different language, we use a different corpus (to train and test KEC) and different criteria (to evaluate the performance of KEC). For details of this corpus and the evaluation method used, see Section 5.5. KEC has been trained and tested on a set of Chinese documents. The experiments show that KEC *successfully* extracts keyphrases from these documents (for details, see Section 5.6). By 'successfully', we mean the algorithm is capable of extracting at least one correct keyphrase from these documents. This is, as far as we know, the first time a keyphrase extraction algorithm has been validated and evaluated on different domains and different languages.

1.4.2 Thesis

The rest of this thesis is organized as follows:

- Chapter 2 discusses related work by other researchers. This chapter reviews a number of keyphrase extraction algorithms, including GenEx and Kea. In addition, we discuss information retrieval, information extraction and summarization, and explain the relationship between them and their relationship to keyphrase extraction. The text processing techniques used in KE (i.e. stemming and part-of-speech tagging) and the learning method used to tune KE (i.e. neural networks) are also discussed.

- Chapter 3 describes the KE algorithm. The algorithm is described using pseudocode and explained with examples and informal English descriptions. In addition, we introduce the training of KE and the normalization of different attributes evaluated in our experiments. This chapter also discusses the differences between KE, GenEx and Kea.

- Chapter 4 presents the experimental results relevant to the KE algorithm. This chapter introduces the corpus used to train and test KE, and the criteria used for evaluating the output keyphrases. We also compare the individual performance of different attributes, the performance of different combinations of attributes and *TF*×*IDF*, the performance of different keyphrase extraction tools, and the performance of KE on different learning methods and different corpora.

- Chapter 5 extends the use of the KE algorithm to Chinese documents. This chapter explains why KE can be adapted to another language and how this is done, and extends KE (to KEC) to extract keyphrases from Chinese documents. We also introduce the corpus and criteria used for evaluating the performance of KEC. This chapter also presents the performance results of KEC on Chinese documents.

- Chapter 6 discusses the conclusions and future work. This chapter summarizes this thesis and the contributions of our research work, and suggests two possible ways to improve the quality of the output keyphrases.

1.5 Summary

This chapter introduces our research work. We have given background information about keyphrase extraction, and discussed some areas to which keyphrases could be beneficial. Terms which will be used in this thesis have been defined in this chapter. An outline of our research work and this thesis have also been provided. The next chapter will discuss related work by other researchers.

CHAPTER TWO
Related Work

2.1 Overview

This chapter summarizes related work by other researchers. We review GenEx, Kea and some recent keyphrase extraction algorithms. A number of research areas related to our work are also discussed: information retrieval (IR), information extraction (IE), summarization, stemming, part-of-speech tagging, and neural networks. Many ideas used in keyphrase extraction are borrowed from IR, e.g. indexing, TF×IDF, recall and precision, so it is worth reviewing IR. IE and keyphrase extraction are closely related. Both of them attempt to extract important information from the document. Summarization and keyphrase extraction are also closely related. Keyphrase extraction algorithms aim at automatically summarizing the content of a document in a few keyphrases, so keyphrase extraction could be seen as a specific subject area of summarization. We also discuss the text processing techniques used in KE: stemming and part-of-speech tagging. Stemming is used to improve the efficiency of KE and evaluate the performance of KE by reducing variants of a word to a single form. Part-of-speech tagging is used to improve the quality of candidate phrases in KE: only adjectives, verbs, nouns, and noun phrases[4] are selected as candidate phrases. In addition, neural networks are introduced in this chapter. This is because KE is tuned by a back-propagation neural network.

[4] In KE, a noun phrase is naively defined as zero, one or two nouns or adjectives followed by a noun or a gerund (for details, see Section 3.3.4).

Section 2.2 reviews a number of keyphrase extraction algorithms. Section 2.3 introduces some useful IR concepts. Section 2.4 discusses IE and the differences between IR and IE. Previous work on summarization and the differences between IR, IE, and summarization are discussed in Section 2.5. Section 2.6 introduces stemming and discusses the Porter, the Lovins, and the iterated Lovins stemming algorithms. Part-of-speech tagging and Eric Brill's part-of-speech tagger are discussed in Section 2.7. Section 2.8 reviews the use of neural networks in different IR systems and introduces neural networks. Section 2.9 concludes this chapter.

2.2 Keyphrase Extraction

This section discusses several keyphrase extraction algorithms. A number of keyphrase extraction algorithms have been proposed [17, 27, 34, 71, 93, 107, 118, 123]. GenEx and Kea are the first two keyphrase extraction algorithms. Despite the fact that they were introduced in the late 1990s, they remain rather important and are reviewed in most of the research papers on keyphrase extraction. We discuss GenEx, Kea and some recent keyphrase extraction algorithms in this section.

2.2.1 GenEx

Turney [106, 107] proposes a keyphrase extraction algorithm called GenEx, which consists of a set of parameterized heuristic rules that are fine-tuned by a genetic algorithm. During training, the genetic algorithm adjusts the rules' parameters to maximize the match between the output keyphrases and the target keyphrases. Table 2.1 shows the parameters used in GenEx. The sample values of these parameters are from [107].

Table 2.1: Parameters used in GenEx

Parameter	Description	Sample Value
NUM_PHRASES	Length of the output list, i.e. the number of keyphrases to be output	10
NUM_WORKING	Length of the working list, i.e. only words ranked higher than this are considered as candidate phrases	50
FACTOR_TWO_ONE	Reward for two-word phrases (or bigrams)	2.33
FACTOR_THREE_ONE	Reward for three-word phrases (or trigrams)	5
MIN_LENGTH_LOW_RANK	Low rank words must be longer than this; if not, they might be removed from the output list	0.9
MIN_RANK_LOW_LENGTH	Short words must be ranked higher than this; if not, they might be removed from the output list	5
FIRST_LOW_THRESH	Definition of early occurrence; words which first occur before this position are rewarded by FIRST_LOW_FACTOR	40
FIRST_HIGH_THRESH	Definition of late occurrence; words which first occur after this position are penalized by FIRST_HIGH_FACTOR	400
FIRST_LOW_FACTOR	Reward for early occurrence	2
FIRST_HIGH_FACTOR	Penalty for late occurrence	0.65
STEM_LENGTH	Maximum characters for fixed length stemming	5
SUPPRESS_PROPER	Flag for suppressing proper nouns	0

There are ten steps in GenEx. We summarize these steps in a number of activities:

1. Stem the words in the input document by truncating them at STEM_LENGTH characters, and select terms and term phrases from the resulting stems.

2. Calculate the score of each term by multiplying the frequency of a term by a factor: the default value of this factor is one, but it is set to FIRST_LOW_FACTOR if this term first occurs before FIRST_LOW_THRESH, and to FIRST_HIGH_FACTOR if this term first occurs after FIRST_HIGH_THRESH.

3. Select the NUM_WORKING highest scoring terms.

4. Calculate the score of each term phrase by multiplying the frequency of a term phrase by the factor used in Step 2 and another factor: the default value of this factor is one, but it is set to FACTOR_TWO_ONE if this term phrase contains two terms, and to FACTOR_THREE_ONE if this term phrase contains three terms.

5. Build a one-to-one relationship between the terms and the term phrases, and use this relationship together with the frequency of phrases to select a list of phrases as output.

6. Perform many post-processing tasks which involve considering the structure and capitalization of phrases, and removing phrases from the output list if they are shorter than MIN_LENGTH_LOW_RANK, ranked lower than MIN_RANK_LOW_LENGTH, or a proper noun when SUPPRESS_PROPER is set to one.

7. Display the top NUM_PHRASES keyphrases in the output list.

GenEx has been trained on a set of journal articles and tested on a different set of journal articles, web pages and email messages. The experiments show that machine learning techniques can be used for the problem of keyphrase extraction and that GenEx generalizes well across collections. While GenEx is trained on a collection of journal articles, it successfully extracts keyphrases from web pages on different topics.

2.2.2 Kea

Frank *et al.* [34] discuss another keyphrase extraction algorithm called Kea, which is based on the naïve Bayes machine learning technique. The basic model of Kea involves two attributes: TF×IDF and *distance* (same as *position*). The standard TF is used, but the IDF is defined differently. Please refer to Section 1.3.5 for the definition of the standard TF. They calculate the IDF of a term *T* in a document *D* by:

$$\text{Kea's IDF} = -\log(\text{no. of documents in collection that contain } T, \text{ excluding } D)^5 \quad (2.1)$$

The naïve Bayes learning method can process numeric (or continuous) attributes by assuming that they have a normal distribution [34]. However, this is not the case for the problem of keyphrase extraction as keyphrases always constitute less than 1% of the document length (and non-keyphrases constitute the remaining 99%). Therefore, both of these attributes have been discretized into a small number of distinct ranges (or intervals), i.e. to convert these numeric attributes into nominal (or discrete) ones, using Fayyad and Irani's discretization method [30] before the learning method is applied. This discretization method recursively partitions an attribute into intervals, minimizes the class entropy at each stage, and stops partitioning when a criterion based on the minimum description length principle is satisfied.

[5] The counters start with one to avoid taking the logarithm of zero.

Kea uses the same set of training and testing documents (excluding the email messages) as in GenEx so that its performance can be directly compared with that of GenEx. The experiments indicate that GenEx and Kea perform at roughly the same level, measured by the average number of matches between author-assigned keyphrases and machine-extracted keyphrases.

Frank *et al.* then extend this model by adding a new attribute, i.e. *keyphrase frequency* (the number of times a phrase occurs as an author-assigned keyphrase throughout the collection, excluding the document that contains this phrase). They find that Kea's performance improves significantly when it is trained on documents that are from the same domain as the document from which keyphrases are extracted. They also argue that Kea can be trained much faster than GenEx in a new domain because of the simple learning technique (i.e. naïve Bayes) employed [34]. Although keyphrase frequency is an interesting concept, the usefulness of this attribute is restricted to a specific domain and is therefore of no use to us. We want KE to be domain-independent and to apply it to heterogeneous documents.

2.2.3 LAKE

D'Avanzo *et al*. [26, 27] propose a keyphrase extraction algorithm called *LAKE*, which is based on the naïve Bayes learning method. It uses two attributes: TF×IDF and *first occurrence* (same as *position*). The main feature that differs it from Kea is that it uses part-of-speech tagging to help to select candidate phrases.

The selection of candidate phrases involves four steps:

1. Tag the input document.

2. Group sequences of words which are considered a single lexical unit together, e.g. 'Christmas' and 'tree' are combined into 'Christmas tree'.

3. Identify all the person, location, and organization names, dates and time, and currency and percentage figures (i.e. named entities) in the document, e.g. ‘London’, ‘December 25 2007’. Named entities will be discussed again in Section 2.4.1.

4. Select candidate phrases from the document if they match one of the many manually predefined linguistics-based patterns, e.g. adjective + noun, and noun + verb + adjective + noun (the symbol ‘+’ denotes ‘followed by’).

The experiments suggest that this algorithm works. Nevertheless, since LAKE uses a different set of training and testing documents, it is not certain if it is better than GenEx and Kea as it is not possible to directly compare their results. Because of this reason, LAKE has not been used as a standard of comparison for evaluating the performance of our algorithm. All the keyphrase extraction tools evaluated in our experiments have been trained and tested on the same corpus (as the one used in GenEx and Kea) so that direct comparison is possible.

2.2.4 KIP

Wu *et al.* [118] introduce a domain-dependent keyphrase extraction system called *KIP*. The system is based on the hypothesis that the more keywords a candidate phrase contains and the more significant these keywords are, the more likely this candidate phrase is a keyphrase.

KIP uses a domain-dependent glossary database (similar to a dictionary) to extract keyphrases from documents. The database consists of two lists: one contains manual keywords and their corresponding weight, and the other contains manual keyphrases and their corresponding weight. KIP selects all the noun phrases in the input document as candidate phrases, and calculates the score of each candidate phrase by the frequency and composition of a phrase (i.e. check if this phrase contains any manual keyword or manual keyphrase stored in the database and obtain the corresponding weight if it does). The system then sorts these phrases in order of score and selects phrases with the highest scores as the output keyphrases.

KIP has been tested on a set of journal and conference papers on information systems, and the experiments show that it works. Nevertheless, since the system uses a different corpus from ours, it is difficult to directly compare its results with ours. Therefore, KIP has not been used as a standard of comparison in our experiments.

2.2.5 Kex

Chen *et al.* [17] propose an algorithm for automatically extracting keyphrases from web pages called *Kex*. The algorithm selects phrases containing at most four terms as candidate phrases. Six attributes are used in Kex: 1) the number of occurrences of a phrase in the web page, 2) the average number of occurrences of a term in the phrase, 3) a formula that involves TF, IDF, and document length, 4) the number of terms in the phrase that occurs in the *meta* tag or the title of the page, 5) the number of terms in the text segments that occur before and after the current text segment, and 6) a score used to discriminate between text segments in different visual styles.

Kex has been trained and tested on a set of web pages from www.msn.com. Human assessors were asked to assign keyphrases to these pages. The experiments confirm that Kex can be used to extract keyphrases from web pages. Nevertheless, since Kex uses a different set of training and testing documents from ours, it is not possible to directly compare its results with ours. Therefore, Kex has not been used as a standard of comparison in our experiments.

2.2.6 KPSpotter

Song *et al.* [93] discuss a keyphrase extraction system called *KPSpotter*. The system can process various formats of input data such as XML, HTML and unstructured text data, and generate an XML file as output. It involves two attributes: TF×IDF and Distance from First Occurrence (same as *position*). These numeric attributes are discretized into ranges using the equal-depth (frequency) partitioning method. The resulting nominal attributes are used to calculate the information gain of each candidate phrase. The candidate phrases are then ranked in order of information gain.

KPSpotter has been trained and tested on a set of abstracts (rather than full documents) of computer science technical reports. The same data have been used to train and test Kea so that the performance of KPSpotter can be directly compared with that of Kea. The experiments show that KPSpotter and Kea give similar results. Nevertheless, since KPSpotter uses a different set of training and testing data (i.e. a collection of abstracts rather than documents), it is not possible to directly compare its results with ours. Therefore, KPSpotter has not been used as a standard of comparison in our experiments.

2.2.7 Kea++

Medelyan and Witten [71] propose a new method of improving the quality of the output keyphrases called *Kea++*. Kea++ is based on Kea, but differs from it in two ways: Kea++ uses a domain-dependent thesaurus and a different set of attributes. Non-descriptors in the document are first replaced by their equivalent descriptors using semantic information about terms and phrases in the thesaurus. Descriptors and non-descriptors are synonyms. Descriptors refer to the 'preferred' terms, and non-descriptors refer to the 'less preferred' terms, e.g. 'love' is a descriptor and 'affection' is a non-descriptor. Candidate phrases are then measured by four attributes: TF×IDF, distance, the length of a candidate phrase in words (same as *number of terms*), and the node degree. The first two attributes are used in Kea. The node degree is the number of thesaurus links that connect a candidate phrase to other candidate phrases.

Kea++ has been tested on a set of documents on food and agriculture. The experiments indicate that Kea++ significantly outperforms Kea. Nevertheless, Kea++ has not been used as a standard of comparison in our experiments. Kea++ uses a controlled vocabulary list and is tied to a specific domain, while KE is a domain-independent algorithm. In addition, Kea++ uses a different set of training and testing documents, so it is not possible to directly compare its results with ours.

2.2.8 W3SS

Zhang *et al.* [123] introduce a new approach to automatic summarization of web sites called *W3SS*. The output summary is based on keywords and keyphrases extracted from a given web site. W3SS uses machine learning techniques to generate this summary, and this involves a number of steps (see Table 2.2).

While keywords and keyphrases are extracted (in Step 4 and 5), it is observed that 40-70% of the keywords and 20-50% of the keyphrases appear somewhere in the home page of a web site. This is because the home page often gives a general idea of what the site is about (this information is likely to be useful for summarizing the whole site), and as we go deeper into the site, web pages tend to be more specific (this information is likely to be too specific and not useful for summarizing the whole site).

W3SS has been tested on a set of web sites. Human assessors are divided into four groups and asked to answer questions about those sites (e.g. the purpose of a site) after 1) they read the generated summaries, 2) they read the manual summaries, 3) they browse only the home pages of those sites, 4) they browse each of those sites for 10 minutes. The experiments show that the group that read the manual summaries give the best results, followed by the group that read the generated summaries, the group that browse each site for a limited time, and the group that browse only the home pages.

Despite being interesting, W3SS has not been used as a standard of comparison in our experiments. All the documents used in our experiments are plain text, i.e. there is no anchor text and special text. The aim of W3SS is also different from ours: W3SS aims at summarizing a collection of web documents (i.e. web site), while KE aims at summarizing a single document.

Table 2.2: Overview of W3SS

Step	Input	Output
1. Follow the links in the home page of a given web site using breadth-first traversal	Web site	Set of web pages
2. Remove all the HTML tags and scripts in these pages	Set of web pages	Set of plain text
3. Use the number of words in a paragraph and the part-of-speech of the words in a paragraph to extract narrative paragraphs from the plain text	Set of plain text	Set of narrative text
4. Use the part-of-speech of a word and the number of occurrences of this word in the narrative text, anchor text (e.g. hyperlinks) and special text (e.g. italic text) to extract keywords	Set of narrative text, anchor text and special text	Set of keywords
5. Use the keywords, the part-of-speech of a phrase and the number of occurrences of this phrase in the narrative text, anchor text and special text to extract keyphrases	Set of narrative text, anchor text, special text and keywords	Set of keyphrases
6. Use the extracted keywords and keyphrases to extract key sentences	Set of narrative text, keywords and keyphrases	Set of key sentences
7. Provide the extracted keywords, keyphrases and key sentences as a summary of this site	Set of keywords, keyphrases and key sentences	Summary

2.3 Information Retrieval

This section discusses two important activities in information retrieval (i.e. indexing and searching), some useful retrieval concepts such as TF×IDF, the vector space model, recall and precision, and the Text Retrieval Conferences (TREC).

Information retrieval is often regarded as being synonymous with document retrieval and text retrieval. It involves the retrieval of documents or texts with information content that is relevant to a user's information need [95]. Library systems, e.g. online public access catalogues (OPAC) [44] and search engines, e.g. Google and Yahoo!, are examples of IR systems. IR consists of two related but different activities: *indexing* and *searching*.

Before indexing and searching are discussed, we would like to give an overview of the IR process (see Figure 2.1):

1. The IR system indexes all the documents in the collection.

2. When the user provides the IR system with a query, it indexes the query, searches the document collection and retrieves the relevant documents.

3. The user might provide feedback on the relevance of the retrieved documents. This information helps to improve the performance of the IR system.

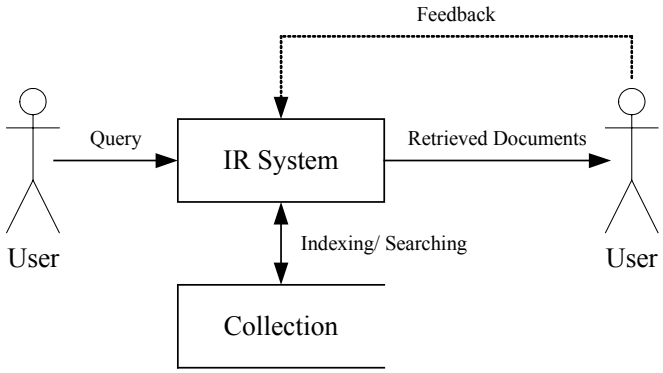

Figure 2.1: Overview of the IR process

2.3.1 Indexing

Indexing refers to the way that documents and *requests* are represented for retrieval purposes. Requests are expressions of a user's information need in natural language, and are translated into queries for the actual searching [95]. A document (or request) text is often represented by a set of index terms. Not all the words in the text are useful for retrieval purposes. Some of them are less important than others and are not generally used for searching. To eliminate these words, a stopword list is used. Unfortunately, the remaining words often have many morphological variants. To solve this problem, a stemming algorithm is used to reduce variants of a word to a single form.

A stopword list contains words with high frequency and little semantic value, e.g. articles, prepositions, and connectives. The use of a stopword list not only eliminates unimportant words, but also reduces the size of the inverted file by 30-50%, and thus improves the efficiency of the IR system [110]. For example, in the TREC collection, there are 33 terms that occur in more than 38.2% of the documents. These terms account for almost 30% of all term occurrences and 11% of the pointers stored in the inverted file. A further 39 terms occur in 24.5-38.2% of the documents and account for another 6.3% of the pointers. The next 63 terms occur in 18.1% of the documents and account for 7.3% of the pointers. In combination, these three groups of terms account for 25% of the pointers in the inverted file [115]. Nevertheless, stopword lists have a problem: the system can hardly retrieve documents containing phrases like 'To be, or not to be'. Therefore, some search engines use a full text representation [2]. A full text representation always demands more computational resources, while a very concise representation might lead to poor retrieval results (i.e. low recall). However, due to recent advances in computer technology (e.g. storage devices have become cheaper), there is a tendency to use almost all the words in a document (or request) text.

A stemming algorithm aims at overcoming the variation of word forms (e.g. singular/ plural forms), which is likely to be encountered in a free text environment [95]. Stemming also helps to reduce the size of the inverted file. For example, after case-folding (i.e. replacing all uppercase characters with lowercase) and stemming the raw documents in the TREC collection, the number of pointers in the inverted file is reduced by 16% and the number of distinct terms by about 40% [115]. Several stemming algorithms will be discussed in more detail in Section 2.6.

2.3.2 Searching

Searching refers to the way that a document collection is examined and the documents in it are retrieved as relevant to the search query. It consists of two important operations: *matching* and *scoring*. Matching establishes what is in common between the document and query representations, normally what terms are shared. Scoring assigns a particular value to the match according to the chosen system function (or retrieval model) [95].

Given sets of terms that characterize a user's information need (i.e. query) and the content of each of the documents in the collection, a best match search involves calculating a score denoting the similarity between the sets of terms for the query and for each document. The documents are then ranked in descending order of score, i.e. the documents at the top of the list are judged to be the best match for the query and are therefore displayed first to the user [95].

The similarity measure consists of two important components: *term weighting* scheme and *similarity coefficient*. The term weighting scheme allocates numerical values to each of the index terms in the document (or query) to reflect their relative importance. The similarity coefficient uses these weights to calculate the overall degree of similarity between a document and a query. The term weighting scheme plays an important role in IR and is probably the most important factor in determining the effectiveness of an IR system [95].

2.3.3 Term Weighting Scheme

Before the term weighting scheme is discussed, we need to explain why statistical methods can be used for retrieval purposes. The number of occurrences of distinct words in natural language documents is always different, and this is useful for content representation. If the frequency were the same, it would be impossible to distinguish between different words using quantitative criteria [90]. In fact, it has long been observed that words occur unevenly in natural language documents. As a result, words can be distinguished by their frequency. In one of Luhn's early papers [65], he suggests that:

"The justification of measuring word significance by use-frequency is based on the fact that a writer normally repeats certain words as he advances or varies his arguments and as he elaborates on an aspect of a subject. This means of emphasis is taken as an indicator of significance."

It is known that when the distinct words in a document are arranged in descending order of frequency, the relationship between the frequency of words and their rank order can be characterized by *Zipf's law* [124]:

$$frequency \times rank \approx \text{constant}$$

Zipf's law states that the frequency of a given word multiplied by the rank order of that word is approximately the same as the frequency of another word multiplied by its rank order. The law has been explained by citing a general 'principle of least effort' which makes it easier for authors to repeat certain words instead of using new and different words. The least-effort principle also accounts for the fact that the most frequent words tend to be short function words (e.g. and, the, etc). The law has been verified by text materials in different areas. Table 2.3 gives an example of such verification from [90]. It has also been reported that the most frequent 20% of the words in natural language documents account for some 70% of word usage [90].

Table 2.3: Verification of Zipf's law [90]

Rank (R)	Word	Frequency (F)	R • (F/ 1,000,000)
1	The	69,971	0.070
2	Of	36,411	0.073
3	And	28,852	0.086
4	To	26,149	0.104
5	A	23,237	0.116
6	In	21,341	0.128
7	That	10,595	0.074
8	Is	10,099	0.081
9	Was	9,816	0.088
10	He	9,543	0.095

Luhn [65] also suggests that neither high nor low frequency words are good content identifiers, and that middle frequency words, which have high 'resolving' power, are presumably the best content identifiers (see Figure 2.2). By resolving power, he means the ability of a word to identify relevant documents and to distinguish them from the nonrelevant[6] ones. As we will see later, non-significant high frequency words can be eliminated by the use of inverse document frequency, and non-significant low frequency words by the use of term frequency.

[6] The information retrieval community often uses 'nonrelevant', instead of 'irrelevant', to refer to documents that are not relevant.

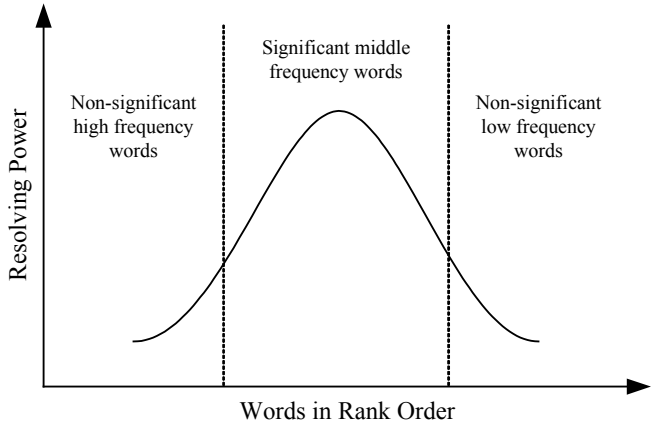

Figure 2.2: Resolving power of significant and non-significant words [90]

One of the most important term weights is term frequency (TF). TF measures the frequency of a term in the document or query, and improves retrieval performance (i.e. higher recall) by rewarding terms that are often mentioned in the document or query.

However, TF alone is not enough to ensure acceptable retrieval performance. When the high frequency terms are not concentrated in a few documents, but are prevalent in the whole collection instead, all documents tend to be retrieved. To solve this problem, another term weight, inverse document frequency (IDF), or collection frequency, is used. IDF measures the rarity of a term across the collection, and improves retrieval performance (i.e. higher precision) by rewarding terms that are concentrated in a few documents of the collection [88].

A common way of measuring term importance is to use the product of TF and IDF, i.e. TF×IDF. Despite the popularity of these weights, they do not have a universal definition.

Salton and Buckley [88] review the use of statistical information for weighting document terms and query terms, and discuss various ways of defining and combining three term weights: TF, IDF, and *length normalization* (LN). LN is used to ensure that short documents have the same chance of being retrieved as the longer ones. A total of 1,800 different term weighting combinations were used in their experiments, and 287 were found to be distinct. They make recommendations on the best combination in different situations. For technical documents (like the ones used in our corpora), they recommend using the *normalized TF* and the standard IDF. The normalized TF is calculated by normalizing the standard TF factor by the maximum TF in the vector[7] [88]:

$$\text{Normalized TF} = 0.5 + 0.5\frac{TF}{\max TF} \tag{2.2}$$

Please refer to Section 1.3.5 for further information about the definition of the standard TF and standard IDF, and to Section 4.2, Section 4.4.7 and Section 5.5 for details of the corpora used in our experiments.

Robertson and Sparck-Jones [85] introduce a simple formula, which combines three term weights: the standard TF, the standard IDF, and the *normalized document length* (NDL), for text retrieval. The NDL is calculated by normalizing the length of a document by the average length of the documents in the collection. The combined weight formula has proved effective through extensive testing during TREC and is easy to apply [85]. The formula is given by:

$$\text{Combined Weight} = \frac{TF \times IDF \times (\mathrm{K}_1 + 1)}{\mathrm{K}_1 \times ((1 - \mathrm{K}_2) + \mathrm{K}_2 \times NDL) + TF} \tag{2.3}$$

where K_1 and K_2 are tuning constants.

[7] The normalization result lies in the range of 0.5 to 1.0.

The constant K_1 (which is usually greater than 0) modifies the extent of the influence of TF, and K_2 (which ranges between 0 and 1) modifies the effect of NDL. Ideally, these constants should be set after systematic trials on the particular document set. In TREC, where heterogeneous sets of documents are used, these constants have been found to be effective when K_1 is set to 2 and K_2 to 0.75. Those values could be used as a starting point for tuning these constants.

2.3.4 Vector Space Model

There are three main types of IR models: the Boolean model (or logical model), the *vector space model* (or vector processing model), and the probabilistic model [2, 95]. Nevertheless, only the vector space model [89] is discussed in this thesis because this is the model which is most related to our work: vectors are used to represent documents in the KE algorithm.

A *model* is an abstraction of a process, so an IR model could be seen as an abstraction of the process of retrieving relevant documents on the basis of a query. The vector space model has been the most influential model in the development of IR [95]. In this model, documents and queries are represented by vectors, in which the *i*-th element denotes the value of the *i*-th term, with the value of each element being determined by the term weighting scheme employed.

Suppose there are *t* terms and *d* documents in an IR system. A document Doc_i can be represented by a vector of weighted terms:

$$\mathrm{Doc_i} = (w_{i1}, w_{ij}, ..., w_{it})$$

where w_{ij} is the weight of the term *j* assigned to the document *i*.

In the vector space model, a document collection can be represented as a matrix of terms where each row of the matrix represents a document and each column represents the assignment of a specific term to the documents of the collection (see Table 2.4) [89].

Table 2.4: Term assignment matrix

	$Term_1$	$Term_j$	…	$Term_t$
Doc_1	w_{11}	w_{1j}	…	w_{1t}
…	…	…	…	
Doc_i	w_{i1}	w_{ij}	…	w_{it}
…	…	…	…	
Doc_d	w_{d1}	w_{dj}	…	w_{dt}

Similarly, a query $Query_k$ can be represented by a term vector:

$$\text{Query}_k = (w_{k1}, w_{kj}, ..., w_{kt})$$

where w_{kj} is the weight of the term j assigned to the query k.

A common way to calculate the degree of similarity between a document vector Doc_i and a query vector $Query_k$ is by using the cosine correlation [88, 89]:

$$\text{Similarity}(\text{Doc}_i, \text{Query}_k) = \frac{\sum_{j=1}^{t} w_{ij} \times w_{kj}}{\sqrt{\sum_{j=1}^{t} (w_{ij})^2 \times \sum_{j=1}^{t} (w_{kj})^2}} \qquad (2.4)$$

This formula measures the cosine of the angle between Doc_i and $Query_k$.

Unlike the Boolean model, terms in the vector space model are not equally weighted; each term is associated with a specific weight which reflects the importance of that term. TF and IDF, which have been discussed in Section 2.3.3, are the two most important term weights in the vector space model.

2.3.5 Evaluation

We need some quantitative methods of evaluating the ability (or effectiveness) of an IR system to retrieve relevant documents and reject nonrelevant documents. The two most common measures of retrieval performance are recall and precision. Recall is the proportion of the relevant documents that are retrieved in a search, while precision is the proportion of the documents retrieved in a search that are relevant.

In principle, an IR system that produces both high recall by retrieving all possibly relevant documents and high precision by rejecting all possibly nonrelevant documents is preferred. The recall function appears to be best served by using broad, high frequency terms that occur in many documents of the collection. These terms are likely to retrieve many documents, including many of the relevant documents. However, the precision function appears to be best served by using narrow, highly specific terms that are likely to isolate the few relevant documents from the mass of nonrelevant documents. In practice, compromises are always made by using terms that are broad enough to produce reasonably high recall without at the same time producing unreasonably low precision [88]. IR systems have been found to operate at no more than 30% recall and 30% precision (in large, realistic test collections), with an increase in one causing a decrease in the other [95].

Because of the trade-off between recall and precision, *F-measure* has been proposed. F-measure is a common way of combining recall and precision. It is the harmonic mean of these measures, i.e. the inverse of the average of the inverses. It penalizes low recall and low precision. The standard F-measure is given by [2]:

$$\text{F-measure} = \frac{1}{0.5 \times (\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}})} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (2.5)$$

F-measure attempts to find the best possible compromise between recall and precision. An IR system will only produce high F-measure if both recall and precision are high [2]. Recall and precision will be discussed again in Section 4.3.3.

2.3.6 TREC

The Text Retrieval Conference (TREC) [103] is one of the most important conferences on IR. The aim of TREC is to encourage research on IR for large text applications by providing large test collections, uniform scoring procedures, and a forum for organizations interested in comparing their results. A test collection is an abstraction of an operational retrieval environment that provides a means of exploring the relative benefits of different IR techniques in a laboratory setting [111]. Recall and precision are the most common measures of performance at TREC. The performance results allow the participants to compare the effectiveness of different IR techniques and to determine how differences between systems affect performance [2].

Before TREC, the evaluation of IR systems was based on small test collections. IR testing was on a relatively small scale, and earlier work tended to use the same test material to maintain comparability. Even by 1990, experiments were still often carried out with collections of 75 requests against 2,000 documents while the test collections had grown slowly from 35 requests against 82 documents, through 225 against 1,400, to 93 against 11,429 [95].

A total of 103 groups participated in TREC 2004. These groups came from 21 different countries and included academic, commercial and government institutions. TREC 2004 consisted of seven areas of focus called tracks: genomics, high accuracy retrieval from documents (HARD), novelty, question answering (QA), robust, terabyte, and web. For details of these tracks, see [111].

The test collections used at TREC consist of three parts: the documents, the *topics* (same as requests), and the *relevance judgements* (an indication of which documents should be retrieved in response to those topics, i.e. sets of relevant documents to those topics) [111].

TREC is designed to evaluate large scale IR systems, and therefore it uses large sets of documents. The primary document sets used at TREC 2004 contain about two gigabytes of text (between 500,000 and 1,000,000 documents), and consist of mainly newspaper and newswire articles, though there are some government documents and computer science abstracts [111]. The document sets used in various tracks, on the other hand, are smaller and larger depending on the needs of the track and the availability of data.

Topics have to be translated into queries before searching can be carried out. Participants in TREC can use any method to create queries from those topic statements. TREC 2004 makes a distinction between two basic query construction techniques: automatic methods and manual methods. An automatic method is a means of deriving a query from a topic statement without any manual intervention, while a manual method is anything else [111].

The relevance judgements could be seen as the 'right answers' to those topics, and are what turns sets of documents and topics into a test collection. Because of the size of the document sets, it is not possible to produce a complete list of relevant documents to each topic by asking the topic author to go through every document and judge its relevance to that topic. Instead, TREC uses a technique called *pooling*: a pool of possible relevant documents is created by taking a sample of documents selected by the various participating IR systems, and is then shown to the human assessors (see Figure 2.3). The sample is constructed by taking the top X documents (usually X=100) retrieved by each system for a given topic and merging them into the pool. The human assessors will only judge documents that are in the pool; those not in the pool are assumed to be nonrelevant and will not be judged. This is a valid sampling technique because the retrieval results are usually ranked, with the documents most likely to be relevant to the topic coming first [41, 111].

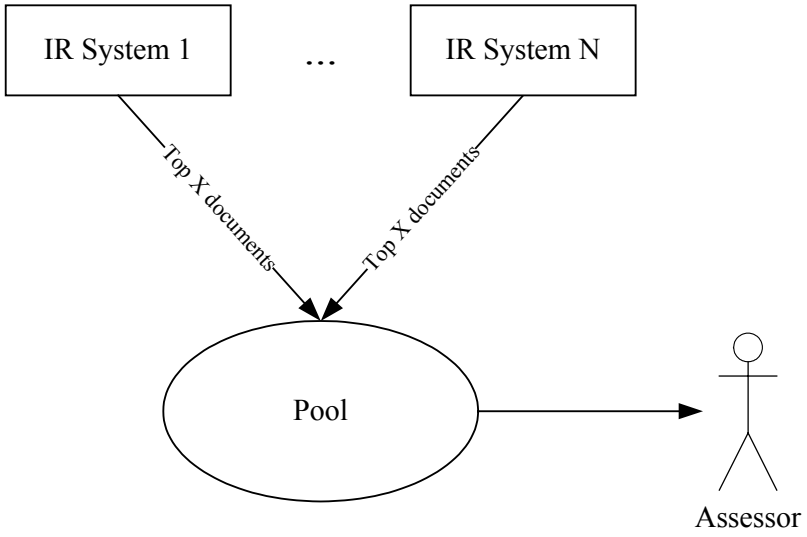

Figure 2.3: Pooling technique

2.4 Information Extraction

This section discusses information extraction, the Message Understanding Conferences (MUC), and the differences between information retrieval and information extraction.

Another related area to keyphrase extraction is *information extraction*. Information extraction involves the identification of pre-specified types of information within a text [95]. An IE system extracts specific information from the document according to some predefined guidelines. These guidelines are always specific to a given topic area or type of text (i.e. domain-dependent). For example, if the topic area is news articles on a terrorist attack, the guidelines might specify that the IE system should identify information about the terrorist organizations involved in the attack, the victims of the attack, the type of the attack, and the date and time of the attack.

A major problem of IE systems is that they are always tied to particular subject domains. Systems that work well in one domain might work badly in other domains, so they have to be tuned before they can be used in a new domain. Recent work, however, shows that machine learning techniques can be used to make IE systems more domain-independent. Tellez-Valero *et al.* [102] build an IE system called *TOPO*, which is based on text classification methods. TOPO uses regular expression analysis to identify candidate text segments, and supervised learning techniques to classify those segments as relevant or irrelevant. The extraction decisions rely on a set of classifiers instead of sophisticated linguistic analysis. Different classifiers have been used for different types of text segments, e.g. name, date and quantity. The experiments indicate that TOPO can be easily adapted to a new domain while maintaining an average F-measure of 72% [102].

2.4.1 MUC

The Message Understanding Conference (MUC) is one of the most important conferences on IE. IE systems have been evaluated with corpora in various topic areas, including naval message narratives (MUCK1, MUCK2; the K in MUCK was dropped from the names of more recent conferences), Latin American terrorism (MUC-3, MUC-4), joint ventures and microelectronics in English and Japanese (MUC-5), and the Wall Street Journal articles (MUC-6) [98, 99]. Recall, precision and F-measure have been used to evaluate the effectiveness of IE systems. There were four extraction tasks at MUC-6 (http://cs.nyu.edu/faculty/grishman/muc6.html): *named entity*, *coreference*, *template element*, and *scenario template* [21, 37, 84, 98].

For example, consider the following text:

"Citigroup on Friday agreed to pay $2bn to settle a class action lawsuit filed by Enron investors who sued the world's largest financial services group for its alleged role in fraudulent deals at the collapsed energy group." – Financial Times, June 10 2005

1. Named entity (NE) involves the identification of all the person, location, and organization names, dates and time, and currency and percentage figures. An NE system should recognize ‘Citigroup’, ‘Friday’, ‘$2bn’ and ‘Enron’ as named entities if the above text is used. NE was the simplest and the most reliable task among the four. 15 systems were tested on this task at MUC-6, and half of the systems achieved over 90% F-measure. The best system scored 96% recall and 97% precision.

2. Coreference (CO) involves the identification of coreferring relations among noun phrases. Noun phrase NP_1 is said to corefer with noun phrase NP_2 if they refer to the same entity [109]. A CO system should link ‘the world’s largest financial services group’ and ‘its’ with ‘Citigroup’ if the above text is used. Seven systems were tested on this task at MUC-6. Most systems achieved approximately the same levels of performance: five of them were in the 51-63% recall range and 62-72% precision range.

3. Template element (TE) is based on NE and CO. It involves the extraction of basic information related to person and organization entities (i.e. NE results) from the document (using CO results). The basic information is presented in a fixed-format, database-like structure. Suppose the organization object ‘Citigroup’ contains a slot called ‘descriptor’, a TE system should extract ‘the world’s largest financial services group’ and associate it with ‘descriptor’ if the above text is used. 11 systems were tested on this task at MUC-6. Almost all the systems achieved over 70% F-measure: four of them were able to achieve recall in the 70-80% range while maintaining precision in the 80-90% range. The best system scored 75% recall and 86% precision.

4. Scenario template (ST) is a difficult IE task. It involves the extraction of pre-specified event information from the document, and relating the event information to the particular person and organization entities involved in the event. In other words, it ties TE entities together into event and relation descriptions. The entities and relations involved in the event are presented in a fixed-format structure. An ST system should be able to answer basic questions such as 'Who agreed to pay how much to whom?' if the above text is used. Nine systems were tested on this task at MUC-6. The levels of performance were similar to those achieved in previous MUCs (40-50% recall range and 60-70% precision range). The best system scored 56% F-measure (47% recall and 70% precision).

2.4.2 Comparison with IR

Information extraction is not information retrieval. IR selects a relevant subset of documents from a larger set, while IE extracts information (often salient facts about pre-specified types of events, entities or relationships) from documents. In short, IR gets sets of relevant documents and IE gets facts out of documents [19, 22, 74, 84]. However, the application of IE is usually preceded by an IR phase, which selects a set of documents relevant to a query. An IR system can therefore be viewed as a combine harvester that brings back potentially useful material from vast fields of raw material. An IE system can then transform the material, refining and reducing it to the germ of the original text [18]. The difference between IR and IE can be illustrated by the following example: we use a search engine to retrieve news articles on recent mergers and acquisitions, and an IE tool to extract the bids and companies involved (from these articles).

2.5 Summarization

This section reviews previous work on summarization, and discusses the differences between IR, IE, and summarization.

Automatic text summarization (*summarization* for short) involves the abstraction of the most important parts of text content [95]. Extracting (mainly *sentence extraction*) and abstracting (or *abstract generation*) are two important activities in summarization [70, 121]. The difference between these activities lies in their output. An extract (i.e. output of extracting) is a summary that consists of material entirely copied from the input document, while an abstract (i.e. output of abstracting) is a summary that contains at least some material that is not present in the document [70]. Sentence extraction involves extracting key sentences from the document. Those sentences often lack coherence because of the frequent occurrence of anaphoric references. Noun phrase NP_1 is said to be the anaphoric antecedent of noun phrase NP_2 if NP_2 depends on NP_1 for interpretation [109]. For example, consider the sentence 'John left his wallet on the table', in which the pronoun 'his' (i.e. NP_2) refers to 'John' (i.e. NP_1). Anaphora and coreference (at MUC) sound similar, but they are different things. For details of their difference, see [109]. Anaphora makes it difficult to generate a coherent abstract by concatenating independent extracted sentences. It is very difficult for readers to understand what those anaphors in the abstract mean. Therefore, abstract generation often involves selecting key sentences and dealing with the anaphors in those sentences before putting the sentences together to form an abstract of that document.

Johnson *et al.* [49] discuss techniques for identifying anaphors to improve the quality of machine-generated abstracts. Sentence selection is guided by a set of rules which involves indictor phrases and information about sentence structure obtained from parsing. The rules are designed to identify sentences which contain non-anaphoric noun phrases and introduce important concepts of the document. The experiments confirm that the important concepts are identified, but the abstracts generated are too long [49].

Purely statistical techniques have proved effective for (single term) indexing in many IR projects, but they appear not good enough for complex information entities such as sentences [95]. Therefore, most of the recent work on summarization is based on a combination of statistical and linguistic methods.

Edmundson [28] describes a sentence extraction algorithm based on statistical and linguistic methods. Four attributes are used in the algorithm: *cue words*, *title words* (i.e. words from the title, subtitles, and headings), *key words* (i.e. high frequency content words), and *sentence location*. Cue words are similar to indicator phrases. They are based on the hypothesis that the probable relevance of a sentence is affected by the presence of pragmatic words such as 'significant', 'impossible', and 'hardly'. The cue words are divided into three categories: *bonus words* – words that are positively relevant (e.g. comparatives and superlatives), *stigma words* – words that are negatively relevant (e.g. anaphoric expressions), and *null words* – words that are irrelevant (e.g. prepositions, pronouns and adjectives). Each sentence in the document is scored by a linear function, which involves the four attributes and their corresponding weight. The experiments indicate that sentence location gives the best individual performance and that the best combination involves cue words, title words, and sentence location [28].

Brandow *et al.* [7] discuss a system called *ANES* which performs domain-independent summarization of news documents. The system has been used to generate summaries from a number of publication types (e.g. newspapers, magazines and newswires) and publication sources. Initially, indicator phrases were planned to be used, but they were abandoned because of their domain-dependent nature. Instead, TF×IDF is used to measure the 'uniqueness' of every word in the document. Words with high TF×IDF values together with other attributes are used to determine which sentences are selected. ANES has been evaluated against another system based on a commercial product called *Searchable Lead*, which summarizes the same articles using only the first portion of the documents. The experiments, however, indicate that the Searchable Lead-based summaries significantly outperform the 'intelligent' ANES summaries [7]. We believe this is mainly because of the nature of the document set. News articles are usually quite well-written, and journalists often put important information at the beginning of their documents.

Kupiec *et al.* [58] develop a trainable sentence extraction program based on statistical and heuristic methods. A Bayesian classification function, which takes each sentence in the document and estimates the probability that it is included in the summary, is used in this program. The summary is generated by ranking the sentences according to their probability. The generated summary is then compared with the corresponding manual summary. Five attributes have been evaluated in the experiments: *sentence length cut-off* (i.e. length of a sentence), *fixed-phrase* (i.e. indicator phrase), *paragraph* (i.e. location of a sentence), *thematic word* (i.e. high frequency content word), and *uppercase word* (i.e. similar to *proper noun*). Paragraph has been found to give the best individual performance, and the best combination involves paragraph, fixed-phrase, and sentence length cut-off [58].

Goldstein *et al.* [36] present an analysis of news article summaries generated by sentence selection. Each sentence in the document is scored by a weighted combination of statistical and linguistic methods, and ranked according to its likelihood of being part of the summary. The statistical methods are adapted from standard IR methods, e.g. TF×IDF. The linguistic methods are derived from an analysis of newswire summaries. Their work shows that the evaluation of summarization systems should take into account both the compression ratios and the characteristics of the document set being used.

Keyphrase extraction has attracted less attention of the summarization community compared with sentence extraction and abstract generation. Nevertheless, it is technically easier than them. It does not have to resolve anaphors and the structure of keyphrases is simple. Training for keyphrase extraction is also easier. Most author-assigned keyphrases can be found in the body of the document, but this is not the case for abstract generation [107]. Most author-supplied abstracts are not composed of sentences that appear in the document, so manual work is needed before training can be carried out.

Although indicator phrases, cue words, and fixed phrases have proved effective in some summarization projects, they rely on detailed knowledge of the corpus's language constructs and are therefore not appropriate for KE. This is because KE is intended for heterogeneous documents.

2.5.1 Comparison with IR and IE

Summarization is similar to indexing (in IR), but it is more difficult. Summarization involves using much fewer words to describe a document than indexing; in other words, it requires a higher level of abstraction. Also, the aim of summarization and IR are different. Summarization attempts to get a shorter text (i.e. summary or keyphrase list) from a longer text (i.e. document), while IR often attempts to use a shorter text (i.e. query) to get a longer text (i.e. relevant document) [94].

The main difficulty in evaluating the output keyphrases is that, unlike IE, summarization is more subjective: different people might have different opinions on what the most important parts of a document are. The evaluation of IE systems, however, is more objective: the extracted information, e.g. the terrorist organizations involved in an attack and the date and time of that attack, is always either correct or not correct. However, Jones and Paynter [51] argue that it is reasonable to use author-assigned keyphrases as a standard of comparison for evaluating machine-extracted keyphrases. This will be discussed in more detail in Section 4.3.1.

2.6 Stemming

This section introduces stemming and discusses three stemming algorithms developed for English documents: the Porter algorithm, the Lovins algorithm, and the iterated Lovins algorithm. The focus of this section is on how these algorithms work.

Documents (and queries) often contain unimportant words (e.g. articles, prepositions, and connectives) and morphological variants (i.e. different forms of words). A stopword list is used to eliminate those unimportant words. A stemming algorithm is then applied to the remaining words to reduce all words with the same root to a single form, which is often done by suffix stripping. The use of stopword lists and stemming algorithms reduces the total number of terms in the IR system, and hence reduces the size and complexity of the data in the system.

Words that appear in documents (and queries) often have various forms, e.g. 'connect', 'connected', 'connecting', 'connection' and 'connections'. 'Connect' and 'connection' will not be considered equivalent unless some sort of language processing is performed on these words. The assumption that two words having the same underlying stem belong to the same conceptual group and should be considered equivalent is obviously an oversimplification [110]. For example, sometimes we might want to distinguish between 'organize' and 'organization'. Sometimes, even words which are essentially equivalent may refer to different things in different contexts. However, it is worth pointing out that suffix stripping programs aim at improving retrieval performance and therefore should not be regarded as a linguistic exercise [78].

Morphological variants generally have similar meanings and can be considered equivalent for retrieval purposes. For this reason, a number of stemming algorithms (or *stemmers*) have been proposed. However, only the Porter, the Lovins, and the iterated Lovins algorithms are discussed in this thesis. This is because the Porter and the Lovins algorithms are the two most common ones developed for English documents [46, 56], and the iterated Lovins algorithm has been used to stem the output keyphrases in KE, GenEx and Kea [34, 107]. Stemming algorithms not only conflate different forms of a word to a single form, but also reduce the size of the inverted file. A smaller inverted file size improves the efficiency of the IR system by saving more storage space and processing time. For retrieval purposes, it usually does not matter whether the stems generated are genuine words or not, provided that different words with the same meaning are conflated to the same form and that words with distinct meanings are kept separate [45].

Although it is possible to carry out stemming by looking the stem of every word up in a (stem) dictionary, this is not practical. This kind of stemming, though simple and linguistically accurate, consumes considerable storage space and computational resources, and is therefore rarely used. Apart from the efficiency concern, it also has another major problem: not all the words can be found in the dictionary, some of them might be too new or too technical.

The automatic removal of suffixes from words, on the other hand, is much more efficient. However, this is not perfect either: the success rate for any suffix stripping program is always less than 100%. For example, if 'work' and 'worker' get conflated, so most probably will 'wand' and 'wander'. The error here is that the 'er' of 'wander' has been treated as a suffix, but it is actually part of the stem. This is known as the *over-stemming* error (i.e. words that should not be conflated are grouped together). Similarly, a suffix stripping program might fail to conflate words with similar meanings to a single stem. For example, 'explain' and 'explanation' might be considered different when in fact they belong to the same conceptual group. This is known as the *under-stemming* error (i.e. words that should be conflated are not grouped together). This explains the difficulty of developing suffix stripping programs: the addition of more rules to increase the performance in one area causes the degradation of performance somewhere else. Unless this is noticed in time, it is easy for the program to become much more complex than is necessary. It is also easy to give undue emphasis on cases which appear to be important but are rare in practice. Since there is no simple way to make these distinctions, we would have to tolerate a certain proportion of errors and assume that they will not degrade retrieval effectiveness too much [45, 78].

2.6.1 The Porter Algorithm

The Porter algorithm uses a suffix list (about 60 different suffixes) with various rules for suffix stripping. It treats complex suffixes as compounds made up of simple suffixes. A word that is to be stemmed is characterized by its *measure*, which is the number of constituent alternating vowel-consonant sequences [78, 95]:

$$[C](VC)^m[V]$$

where C is a list of consonants, V is a list of vowels, and m is the measure of this word (square brackets indicate optionality). For example, 1) the measure of 'by' is zero, 2) the measure of 'trees' is one, 3) the measure of 'private' is two [78].

Suffixes are removed in five steps: Step 1 deals with plurals and past participles, Step 2 to 4 deal with the removal of suffixes, and Step 5 deals with tidying up. Each step consists of a number of rules, which are given in the following form [78]:

$$(\text{condition})\ S1 \rightarrow S2$$

This means that if a word ends with suffix *S1* and if the stem before *S1* meets the given *condition* (which usually involves the value of measure), *S1* will be replaced by *S2*. For example, consider the following rule [78]:

$$(m > 1)\ \text{EMENT} \rightarrow$$

This rule will, for example, map 'replacement' to 'replac' because *S1* is 'ement', *S2* is null, and the measure of 'replac' is two.

The remaining stem will then be passed to the next step. If a word fails to satisfy the condition of a rule, it will be tested by the next rule (or passed to the next step, if there is no more rule in that step). However, if a word meets the condition of several rules, only one rule will be obeyed, and this will be the one with the longest matching suffix for that word [78]. Table 2.5 shows how 'generalizations' is stripped to 'gener' using the Porter algorithm.

Table 2.5: The Porter algorithm

Step	Rule	Stem
0	–	Generalizations
1	$\text{S} \rightarrow$	Generalization
2	$(m > 0)\ \text{IZATION} \rightarrow \text{IZE}$	Generalize
3	$(m > 0)\ \text{ALIZE} \rightarrow \text{AL}$	General
4	$(m > 1)\ \text{AL} \rightarrow$	Gener
5	Does not satisfy any rule	Gener

The experimental results show that the Porter algorithm performs slightly better than a much more elaborate system used for IR research [78]. The effectiveness of this algorithm has also been confirmed by other researchers [46, 56].

The Porter algorithm is probably the most widely used stemmer in IR research [2, 45, 95]. There are several reasons for this: it is fast, conceptually simple, and works at least as well as other complex algorithms [78, 95]. Despite its popularity, there is no linguistic basis for this algorithm. It is simply based on the observation that the use of measure helps to decide whether it is wise to remove a suffix [78].

2.6.2 The Lovins Algorithm

The Lovins algorithm uses a much longer suffix list (about 290 different suffixes) with various rules for suffix stripping. Suffixes are removed in two steps: Step 1 deals with the removal of suffixes, and Step 2 deals with the ending of the remaining stems.

The algorithm involves 294 *endings*, 29 *conditions* and 35 *transformation rules*. A complete list of them can be found in [63, 79]. Each ending (similar to *S1* in the Porter algorithm) is associated with one condition (similar to condition in the Porter algorithm). Two examples of these endings are shown below [79]:

.09.
ationally *B*
.07.
ionally *A*

The endings are grouped by suffix length (i.e. *.09.* and *.07.*), from 11 characters down to one. Each ending is followed by a condition code (i.e. *B* and *A*).

The conditions associated with the above endings are shown below [79]:

A – No restrictions on stem[8]
B – Minimum stem length = 3

In the first step, the longest ending which satisfies the associated condition is found and removed. For example, 'nationally' has the ending 'ationally' which is associated with condition *B*. If 'ationally' is removed, it will leave a stem of length one. This violates the condition and is therefore rejected. However, 'nationally' also has the ending 'ionally' which is associated with condition *A*. Condition *A* has no restriction on the stem length, so 'ionally' is removed, leaving 'nat' [79].

In the second step, the transformation rules are applied to the ending of the remaining stem. This step is carried out no matter whether a suffix is removed in the first step. The transformation rules handle features such as letter undoubling (e.g. remove the last 't' in 'sitt' which is the stem after Step 1: 'sitting' → 'sitt' → 'sit'), irregular plurals (e.g. matrix/ matrices), and English morphological oddities caused by the behaviour of Latin verbs of the second conjugation (e.g. assume/ assumption, and commit/ commission) [79]. Table 2.6 shows how 'recursive' is stripped to 'recur' using the Lovins algorithm.

Table 2.6: The Lovins algorithm

Step	Rule	Stem
0	–	Recursive
1	*.03.* ive *A* *A* – No restrictions on stem	Recurs
2	urs → ur	Recur

[8] There is actually an implicit assumption in all conditions, including condition *A*: the minimum stem length is two.

The Lovins algorithm, which was introduced in 1968, was the first stemming algorithm. It was innovative and remarkable for its time and had a strong influence on later work in this area. However, it has also been criticized for its poor performance on short words (or words with short stems) and for failing to include some common endings in its ending list (e.g. 'ements' and 'ents', though their singular form 'ement' and 'ent' are included). The algorithm is larger than the Porter algorithm because of the extensive ending list employed. However, because of this list, the algorithm is faster; it takes only two steps to remove suffixes from words (compared with five in the Porter algorithm) [79].

2.6.3 The Iterated Lovins Algorithm

It has been reported that aggressive stemming is better for keyphrase extraction than conservative stemming [106-108]. Aggressive stemming is more likely to map two words to the same stem, but it is also more likely to make over-stemming errors. The Lovins algorithm is more aggressive than the Porter algorithm, and the iterated Lovins algorithm is more aggressive than the Lovins algorithm. The iterated Lovins algorithm has also been used for stemming in GenEx and Kea [34, 107]. We therefore use this algorithm to stem the input document and the output keyphrases in KE. Basically, there is nothing new about this algorithm; it just repeatedly applies the Lovins algorithm to a given word until it stops changing.

2.7 Part-of-Speech Tagging

This section introduces part-of-speech tagging and discusses Eric Brill's part-of-speech tagger. The focus of this section is on how this tagger works.

Part-of-speech tagging involves choosing the most likely sequence of syntactic categories for the words in a sentence [1]. A typical set of (syntactic) tags is the Penn Treebank tagset (see Table 2.7). It contains 36 tags, but only adjectives (JJ, JJR, JJS), verbs (VB, VBD, VBG, VBN, VBP, VBZ) and nouns (NN, NNS, NNP, NNPS) are used in KE.

Table 2.7: The Penn Treebank tagset [1]

CC	Coordinating conjunction	PP$	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Comparative adverb
EX	Existential there	RBS	Superlative adverb
FW	Foreign word	RP	Particle
IN	Preposition/ subordinating conjunction	SYM	Symbol (mathematical/ scientific)
JJ	Adjective	TO	To
JJR	Comparative adjective	UH	Interjection
JJS	Superlative adjective	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund/ present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non-third person, present
NNP	Proper noun, singular	VBZ	Verb, third person, present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	Wh-pronoun
POS	Possessive ending	WPZ	Possessive wh-pronoun
PRP	Personal pronoun	WRB	Wh-adverb

2.7.1 Eric Brill's Part-of-Speech Tagger

Brill [9] proposes a simple rule-based part-of-speech tagger. Although most part-of-speech taggers are based on statistical techniques, Brill's is based on rules. This tagger works as well as stochastic (or statistical) taggers, but it is simpler and requires less stored information. That is why we use it to select candidate phrases in the KE algorithm. Brill's tagger initially tags by assigning each word its most likely tag, which is estimated by examining a large tagged corpus (i.e. the training corpus) without regard to context. It then uses a smaller tagged corpus (i.e. the patch corpus) to recognize and remedy its weaknesses and improve its performance.

Brill uses the Brown Corpus [33] in his work. The corpus is divided into three parts: training, patch, and testing. The training corpus is used to train the tagger, the patch corpus is used to improve the performance of the tagger, and the testing corpus is used to test the tagger.

The initial tagger has two built-in procedures to improve performance; neither of them makes use of contextual information [9]:

- To tag words which are capitalized and not found in the training corpus as proper nouns.

- To tag words not seen in the training corpus by assigning such words the tag most common for words ending in the same three letters. For example, 'blahblahous' will be tagged as adjective because adjective is the most common tag for words ending in 'ous'. This information is derived from the training corpus.

The tagger then acquires patches to reduce its error rate. After the initial tagger is trained, it is used to tag the patch corpus. A list of tagging errors is generated by comparing the output of the tagger with the correct tagging of the patch corpus. This list consists of triples $<tag_a, tag_b, number>$ which involve the number of times the tagger mistagged a word as tag_a when it should have been tagged as tag_b in the patch corpus. For each error triple, the tagger determines which template from the pre-specified set of patch templates results in the greatest improvement. The patch templates are given by [9]:

Change tag *a* to tag *b* when:

- The preceding (or following) word is tagged as *z*.

- The word two before (or after) is tagged as *z*.

- One of the two preceding (or following) words is tagged as *z*.

- One of the three preceding (or following) words is tagged as *z*.

- The preceding word is tagged as *z* and the following word is tagged as *w*.

- The preceding (or following) word is tagged as *z* and the word two before (or after) is tagged as *w*.

- The current word is (or is not) capitalized.

- The previous word is (or is not) capitalized.

For each patch, the reduction in error caused by applying that patch to the patch corpus is calculated. The patch which results in the greatest error reduction is added to the list of patches. After that, the patch is applied in order to improve the tagging of the patch corpus, and the patch acquisition procedure continues.

The experiments indicate that Brill's tagger performs as well as stochastic taggers. Nevertheless, it has two main advantages over those taggers [8, 9]:

- The tagger is much more portable. It is rather difficult for stochastic taggers to transfer many of the higher level procedures used to improve performance from one tag set, corpus genre or language to another. This rule-based tagger, however, can do that fairly easily.

- The tagger does not require large tables of statistics. In a stochastic tagger, tens of thousands of lines of statistical information are often needed to capture contextual information. However, in this rule-based tagger, contextual information is captured in fewer than 80 rules. This not only makes the tagger smaller and simpler, but also makes it easier to find and implement improvements to the tagger. Contextual information is expressed in a more compact and understandable form compared with the information hidden in those large tables of contextual probabilities.

2.8 Neural Networks

This section reviews the use of neural networks in different IR systems and introduces neural networks.

Many machine learning methods have been proposed [72]. GenEx uses a genetic algorithm to distinguish keyphrases from non-keyphrases [107], Kea, LAKE and Kea++ use the naïve Bayes techniques [27, 34, 71], Kex uses a logistic regression model [17], and W3SS uses a decision tree learning method [123]. Please refer to Section 2.2 for details of these keyphrase extraction algorithms.

KE is tuned by a standard back-propagation neural network (for details, see Section 4.4.1). Neural networks are used because they provide a simple black box for pattern recognition [6, 83, 86] and have proved useful in a number of IR projects [48, 113, 119]. Cunningham *et al.* [23] provide an excellent review of the application of neural networks to IR. For further information about the application of other machine learning techniques to IR, see [16, 24].

Wilkinson and Hingston [113] implement an IR system based on neural networks. The system takes a query as input, compares all the documents in the collection with the query, and returns the relevant ones as output. Neural networks are used to identify the relationships between the input query and the output documents. In their system, each query term is associated with an input unit, each of the set of terms representing all the documents in the collection is associated with a hidden unit, and each document is associated with an output unit. The experiments indicate that neural networks can be used as an alterative IR model and that standard IR techniques (e.g. the cosine correlation) can be used in this model [113].

Jo [48] describes a keyword extraction algorithm which involves six attributes: TF, IDF, *inverted term frequency* (ITF), title, *first sentence*, and *last sentence*. The output of the algorithm is a set of important words (rather than phrases) automatically extracted from the body of the document. ITF is the number of occurrences of a term throughout the collection. First sentence is a flag that indicates if a term occurs in the first sentence of the document. Similarly, last sentence is a flag that indicates if a term occurs in the last sentence of the document. First sentence and last sentence have a similar effect to *position*. The algorithm has been tuned by a back-propagation neural network and tested on a collection of news articles. The experiments indicate that the resulting algorithm performs better than two term weighting equations based on TF×IDF [48].

You *et al.* [119] build an IR system that automatically retrieves hot topics from a bulletin board system (BBS). The system has been trained to classify topics as hot or not hot by different machine learning methods, including neural networks and the naïve Bayes techniques. The experiments show that back-propagation neural networks give the best performance result [119].

As mentioned earlier, KE is tuned by a standard back-propagation neural network. Therefore, we think it is worth introducing neural networks. The idea underlying a neural network is simple. The network has to learn how to make predictions [81]. During training, we provide it with a set of input vectors and the corresponding target output vectors. Each input value and target output value is associated with a neuron (or *unit*). The set of input neurons forms the input layer and the set of output neurons forms the output layer. The number of input units and output units are constrained by the training examples [72].

Typically, there is another set of neurons between the input layer and the output layer called the *hidden* layer. It is called 'hidden' because the output of these hidden units is available only within the network and is not available as part of the global network output [72]. It is possible to have more than one hidden layer in a neural network, but one hidden layer is adequate for most applications [29, 86]. The number of hidden units is variable and affects the generalization performance of the network [54, 59]. The choice depends on a number of factors such as the number of training examples and the complexity of the classification task that the network is trying to learn. If the number of hidden units is too small, the network will not have enough power to learn from the training data (i.e. underfitting) and will likely produce high training error and high generalization error. However, if the number of hidden units is too large, the network will tend to memorize the training data (i.e. overfitting) and produce low training error but high generalization error. For most problems, there is only one way to find the best number of hidden units: try many networks with different number of hidden units and find the best network [5, 40, 80].

Each unit is connected with other units by means of communication links, each with an associated *weight*. The weights may be positive or negative [29, 86]. Typically, each hidden unit is connected with all the input units, and each output unit is connected with all the hidden units. This is known as a *fully connected* neural network.

2.9 Summary

This chapter summarizes related work by other researchers. A number of keyphrase extraction algorithms have been reviewed. In addition, we have discussed IR, IE and summarization, and compared their differences. The text processing techniques used in KE (i.e. stemming and part-of-speech tagging) and the learning method used to tune KE (i.e. neural networks) have also been introduced. The next chapter will describe the KE algorithm.

CHAPTER THREE

The KE Algorithm

3.1 Overview

This chapter describes the KE algorithm. KE is based on GenEx and Kea, but differs from them in several ways: it uses a combination of statistical and text processing techniques, a different set of attributes, and a different machine learning method to extract keyphrases from documents. For details of the differences between KE, GenEx and Kea, see Section 3.5. Part-of-speech tagging, which is a useful text processing technique, has been used to select only adjectives, verbs, nouns, and noun phrases as candidate phrases. Five attributes have been found useful for keyphrase extraction in our experiments and are used in KE: *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. Descriptions of these attributes can be found in Section 1.3. This chapter is mostly concerned with the description of KE. We explain in detail how KE works using pseudocode. The training of KE using a neural network is also discussed. After training, KE can be used to extract keyphrases from new documents.

Section 3.2 gives an overview of KE. A high-level description of KE using pseudocode is provided in Section 3.3. Section 3.4 discusses how KE is trained and why attributes evaluated in our experiments have to be normalized and how this is done. Section 3.5 compares KE with GenEx and Kea. Section 3.6 concludes this chapter.

3.2 Overview of KE

When a user provides KE with a document, the title of the document and the desired number of output keyphrases, the title is stemmed and the document is tagged and stemmed. We use the iterated Lovins stemmer to stem the input title and the input document (for efficiency purposes) and the output keyphrases (for evaluation purposes), and Eric Brill's part-of-speech tagger to tag the input document. The iterated Lovins stemmer is chosen because 1) it has been used for stemming in GenEx and Kea [34, 107] 2) aggressive stemming is better for keyphrase extraction than conservative stemming [106-108]. Eric Brill's part-of-speech tagger is selected because it works as well as statistical taggers but it is simpler and requires less stored information [9]. Please refer to Section 2.6 and Section 4.3.2 for further information about the stemmer, and to Section 2.7 for the tagger. Unimportant words in the document are filtered out by using the stopword list and selecting only adjectives, verbs, nouns, and noun phrases as candidate phrases.

The resulting title and document are then used to identify keyphrases. There are seven steps in KE (see the Gantt chart in Figure 3.1). These steps are discussed in detail using pseudocode in Section 3.3. In addition, a formal description of these steps using the Z notation is provided in the Appendix. Step 1 and 2 of the algorithm are conceptually independent of Step 3 and 4, so they can be carried out at the same time if adequate resources are available. Table 3.1 summarizes the input and output of each step.

After going through the above steps, KE provides a list of keyphrases as output. A phrase which is nearer the top of the list is more likely to be a keyphrase. No keyphrase should appear more than once in the list and this is reinforced by Step 6 of the algorithm. This is because identical keyphrases add no value to the summary generated.

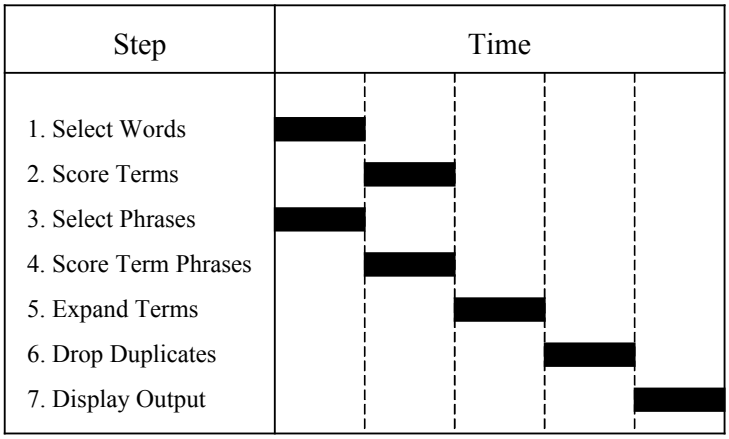

Figure 3.1: Overview of the KE algorithm

Table 3.1: Input and output of each step in KE

Step	Input	Output
1. Select Words	Input document	List of words
2. Score Terms	List of words (output of Step 1)	List of terms ordered by the scores calculated in this step
3. Select Phrases	Input document	List of phrases
4. Score Term Phrases	List of phrases (output of Step 3)	List of term phrases ordered by the scores calculated in this step
5. Expand Terms	List of terms and list of term phrases (output of Step 2 and 4)	List of term phrases ordered by the scores calculated in Step 2
6. Drop Duplicates	List of term phrases (output of Step 5)	List of term phrases
7. Display Output	List of term phrases (output of Step 6)	List of keyphrases

As we will see, terms are used for ranking purposes (see Section 3.3.6). This is because previous work suggests that it is generally preferable to represent documents and measure the importance of each representation element using single terms rather than term phrases [88]. Term phrases, on the other hand, are used for output purposes (see Section 3.3.8). This is because documents are summarized by a set of phrases, not words.

3.3 Description of KE

This section gives a high-level description of the KE algorithm using pseudocode. Table 3.2 shows the variables used in the pseudocode described in the following sub-sections. The meaning of these variables will become clear as the algorithm is described.

Table 3.2: Variables used in the pseudocode for KE

Variable	Description
lcp	List of phrases corresponding to the term phrases in *lutp*
lctp	List of term phrases containing the terms in *lt*
lk	List of keyphrases
lltp	List of term phrases linked to the terms in *lt*
lp	List of phrases selected from the input document
ls	List of stems generated from *lw*
lsp	List of stem phrases generated from *lp*
lt	List of terms generated from *ls*
ltp	List of term phrases generated from *lsp*
lup	List of unique phrases generated from *lcp*
lutp	List of unique term phrases generated from *lltp*
lw	List of words selected from the input document
sid	Stemmed input document
sit	Stemmed input title
tid	Tagged input document

3.3.1 Attributes

The selection of relevant attributes is probably the most important factor in determining the effectiveness of a keyphrase extraction algorithm. If we do not select relevant attributes (i.e. attributes which together convey enough information to make learning tractable), any attempt to apply machine learning techniques is likely to fail. This is why Witten and Frank [114] argue that the choice of a learning method is usually much less important than coming up with a set of relevant attributes.

Many attributes have been considered, e.g. the frequency of a term, the length of a document, the position of a term in the document, the number of characters in a term, the number of occurrences of a term throughout the collection, etc (for details, see Section 4.4.2). However, only five attributes have been found useful for keyphrase extraction in our experiments: *TF×IDF*, *position*, *title*, *proper noun*, and *number of terms* (for details, see Section 4.4). Please refer to Section 1.3 for the definitions of these attributes.

In KE, a document is represented by a set of terms and a set of term phrases. Terms are characterized by four attributes: *TF×IDF*, *position*, *title*, and *proper noun*. *Number of terms* is not appropriate because it is always one when it comes to single terms and thus fails to discriminate between different terms. Term phrases are also characterized by four attributes: *TF×IDF*, *position*, *title*, and *number of terms*. *Proper noun* is not included because preliminary results show that the additional utilization of this attribute does not make much improvement on the performance of KE[9].

We have implemented and tested different combinations of *TF×IDF*. The standard TF and Kea's IDF have been found to give the best performance (for details, see Section 4.4.4), so they are used to define TF and IDF respectively.

[9] A proper noun phrase was defined as a phrase containing at most three consecutive proper nouns.

3.3.2 Selecting Words

Step 1 involves the selection of all the words which have been tagged as adjective, verb and noun, and are not included in the stopword list.

As mentioned in Section 2.7, there are 36 part-of-speech tags in the Penn Treebank tagset. Nevertheless, we are only interested in adjectives (JJ, JJR, JJS), verbs (VB, VBD, VBG, VBN, VBP, VBZ) and nouns (NN, NNS, NNP, NNPS) (see Table 3.3).

Table 3.3: Tags used in KE

JJ	Adjective	VB	Verb, base form
JJR	Comparative adjective	VBD	Verb, past tense
JJS	Superlative adjective	VBG	Verb, gerund/ present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non-3s, present
NNP	Proper noun, singular	VBZ	Verb, 3s, present
NNPS	Proper noun, plural		

A stopword list contains words with high frequency and little semantic value (for details, see Section 1.3.1). The stopword list used in KE contains 17 common verbs, which are basically the various forms of ‘be’, ‘do’, and ‘have’: ‘be’, ‘were’, ‘was’, ‘being’, ‘am’, ‘been’, ‘are’, ‘is’, ‘do’, ‘did’, ‘doing’, ‘done’, ‘does’, ‘have’, ‘had’, ‘having’, and ‘has’.

It is unlikely that adjectives and verbs will be output, but they are still being considered. They help to boost the score of their noun form (provided their stems are the same as the noun’s) and therefore increase the likelihood that it will be output. Suppose that the input document contains ‘compute’, ‘computation’, and ‘computational’. Even though it is unlikely that ‘compute’ and ‘computational’ will be output, they help to boost the score of the stem ‘comput’, and therefore increase the likelihood that ‘computation’ will be output.

Here is the pseudocode for this step:

```
Step:      Select Words
Input:     Tagged input document, tid
Output:    List of words, lw
Method:
1          FOR each word w in tid
2           IF the part-of-speech of w is adjective, verb or noun
3            IF w is not in the stopword list
4             Add w to lw
5            END IF
6           END IF
7          END FOR
```

The variable *w* in the 'FOR each' loop takes on a new value from *tid* for each iteration. The first time through the loop, it is the first word in *tid*; the second time, it is the second word in *tid*; the last time, it is the last word in *tid*.

3.3.3 Scoring Terms

Step 2 involves six closely related tasks:

1. Stem the selected words

2. Detect equivalent stems

3. Delete terms that occur only once in the document

4. Calculate the *TF×IDF* (using the standard TF and Kea's IDF), *position*, *title* and *proper noun* of each term

5. Assign a score to each term based on these attributes

6. Sort the terms in descending order of score (if two terms have the same score, they are ranked in ascending order of *position*)

Multiple occurrences of a given stem (i.e. words with the same stem) are combined into a single term in Task 2. For example, 'mathematics' and 'mathematician' are combined into 'mathemat' if they occur in the input document (see Figure 3.2).

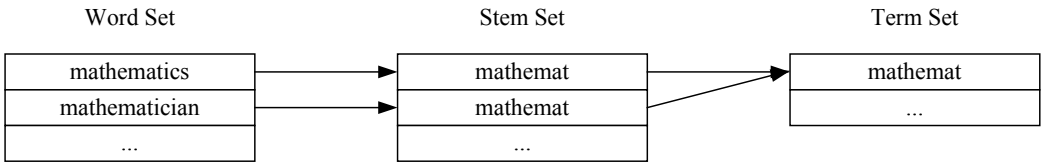

Figure 3.2: Detecting equivalent stems

We noticed in our experiments that some terms had the same score. Therefore, we tried to sort these terms according to their attribute values, such as *TF*×*IDF* and *position*. The experiments show that KE gives better performance results if two terms having the same score are ranked in ascending order of *position*. Terms with smaller *position* values are preferred because their first appearance in the document precedes that of terms with larger *position* values.

Here is the pseudocode for this step:

```
Step:     Score Terms
Input:    List of words, lw
Output:   List of terms, lt, ordered by the scores calculated in this
          step
Method:
1         // Get a list of stems
2         FOR each word w in lw
3          Add the stem of w to ls
4         END FOR
5
6         // Get a list of unique stems
7         FOR each stem s in ls
8          IF s is not in lt
9           Add s to lt
10         END IF
11        END FOR
12
13        // Calculate TF
14        FOR each term t in lt
15         Set TF to the number of occurrences of t in sid
```

```
16        END FOR
17
18        // Delete terms that occur only once
19        FOR each term t in lt
20         IF the TF of t is 1
21          Remove t from lt
22         END IF
23        END FOR
24
25        // Calculate IDF, position, title, proper noun, and score
26        FOR each term t in lt
27
28         // Calculate IDF
29         Set n to the number of documents in the collection that
          contain t (excluding the input document)
30         Set IDF to -log(n+1)
31
32         // Calculate position
33         Set position to the number of stems before the first
          appearance of t in sid
34
35         // Calculate title
36         IF t occurs in sit
37          Set title to 1
38         ELSE
39          Set title to 0
40         END IF
41
42         // Calculate proper noun
43         IF the part-of-speech of t is proper noun
44          Set proper noun to 1
45         ELSE
46          Set proper noun to 0
47         END IF
48
49         // Calculate score
50         Calculate the score of t based on TF×IDF, position, title,
          and proper noun
51
52        END FOR
53
54        // Sort the terms
55        Rank the terms in lt in descending order of score
56        IF two terms have the same score
57         Rank these terms in ascending order of position
58        END IF
```

The code from line 1 to 4 deals with Task 1, line 6 to 11 deals with Task 2, line 13 to 23 deals with Task 3, line 28 to 47 deals with Task 4, line 49 to 50 deals with Task 5, and line 54 to 58 deals with Task 6. The 'END FOR' in line 52 corresponds to the 'FOR' in line 26.

3.3.4 Selecting Phrases

Step 3 involves the selection of all the noun phrases in the document. Like KE, D'Avanzo *et al*. [26, 27] use a part-of-speech tagger to select candidate phrases in the LAKE algorithm (see Section 2.2.3): candidate phrases are selected if they match one of the many manually predefined linguistics-based patterns, e.g. adjective + noun, and noun + verb + adjective + noun (the symbol '+' denotes 'followed by'). Nevertheless, we believe this could be simplified by selecting only noun phrases from the document. This is because almost all the keyphrases are noun phrases and they normally contain less than four words and match the following pattern [108]:

$$(\text{NN} \mid \text{NNS} \mid \text{NNP} \mid \text{NNPS} \mid \text{JJ})^{0..2}\ (\text{NN} \mid \text{NNS} \mid \text{NNP} \mid \text{NNPS} \mid \text{VBG})$$

This pattern means zero, one or two nouns or adjectives (NN | NNS | NNP | NNPS | JJ) followed by a noun or a gerund (NN | NNS | NNP | NNPS | VBG).

A two-word phrase that matches the above pattern has at most three candidates. For example, 1) prime (JJ) number (NN) has two: number and prime number, 2) important (JJ) algorithm (NN) has two: algorithm and important algorithm, 3) decision (NN) making (VBG) has three: decision, making and decision making.

A three-word phrase that matches the above pattern has at most six candidates. For example, 1) large (JJ) prime (JJ) numbers (NNS) has three: numbers, prime numbers and large prime numbers, 2) integer (NN) factorization (NN) algorithm (NN) has six: integer, factorization, algorithm, integer factorization, factorization algorithm and integer factorization algorithm.

Here is the pseudocode for this step:

```
Step:      Select Phrases
Input:     Tagged input document, tid
Output:    List of phrases, lp
Method:
1          FOR each word w in tid
2           IF the part-of-speech of w is noun or gerund
3            Add w to lp
4            Set v to the word right before w
5            IF the part-of-speech of v is adjective or noun
6             Add v followed by w to lp
7             Set u to the word right before v
8             IF the part-of-speech of u is adjective or noun
9              Add u followed by v followed by w to lp
10            END IF
11           END IF
12          END IF
13         END FOR
```

3.3.5 Scoring Term Phrases

Similar to Step 2, Step 4 involves six closely related tasks:

1. Stem the selected phrases

2. Detect equivalent stem phrases

3. Delete term phrases that occur only once in the document

4. Calculate the *TF×IDF* (using the standard TF and Kea's IDF), *position*, *title*, and *number of terms* of each term phrase

5. Assign a score to each term phrase based on these attributes

6. Sort the term phrases in descending order of score (if two term phrases have the same score, they are ranked in ascending order of *position* followed by descending order of *number of terms*)

This step is very similar to Step 2, so readers could see Section 3.3.3 for reference.

We noticed in our experiments that some term phrases had the same score. Therefore, we tried to sort these term phrases according to their attribute values such as *TF*×*IDF*, *position* and *number of terms*. The experiments show that KE gives better performance results if two term phrases having the same score are ranked in ascending order of *position* followed by descending order of *number of terms*. Term phrases with smaller *position* values are preferred because their first appearance in the document precedes that of term phrases with larger *position* values. Longer term phrases are preferred because they generally provide more information than shorter term phrases.

Here is the pseudocode for this step:

```
Step:     Score Term Phrases
Input:    List of phrases, lp
Output:   List of term phrases, ltp, ordered by the scores calculated
          in this step
Method:
1         // Get a list of stem phrases
2         FOR each phrase p in lp
3          Add the stem of p to lsp
4         END FOR
5
6         // Get a list of unique stem phrases
7         FOR each stem phrase sp in lsp
8          IF sp is not in ltp
9           Add sp to ltp
10         END IF
11        END FOR
12
13        // Calculate TF
14        FOR each term phrase tp in ltp
15         Set TF to the number of occurrences of tp in sid
16        END FOR
17
18        // Delete term phrases that occur only once
19        FOR each term phrase tp in ltp
20         IF the TF of tp is 1
```

```
21          Remove tp from ltp
22         END IF
23        END FOR
24
25        // Calculate IDF, position, title, number of terms, and
          score
26        FOR each term phrase tp in ltp
27
28         // Calculate IDF
29         Set n to the number of documents in the collection that
          contain tp (excluding the input document)
30         Set IDF to -log(n+1)
31
32         // Calculate position
33         Set position to the number of stems before the first
          appearance of tp in sid
34
35         // Calculate title
36         IF tp occurs in sit
37          Set title to 1
38         ELSE
39          Set title to 0
40         END IF
41
42         // Calculate number of terms
43         Set number of terms to the number of terms in tp
44
45         // Calculate score
46         Calculate the score of tp based on TF×IDF, position,
          title, and number of terms
47
48        END FOR
49
50        // Sort the term phrases
51        Rank the term phrases in ltp in descending order of score
52        IF two term phrases have the same score
53         Rank these term phrases in ascending order of position
54         IF two term phrases have the same position
55          Rank these term phrases in descending order of number of
          terms
56         END IF
57        END IF
```

The code from line 1 to 4 deals with Task 1, line 6 to 11 deals with Task 2, line 13 to 23 deals with Task 3, line 28 to 43 deals with Task 4, line 45 to 46 deals with Task 5, and line 50 to 57 deals with Task 6. The 'END FOR' in line 48 corresponds to the 'FOR' in line 26.

Because of the way how *position* is calculated (see line 33), it is possible that two term phrases have the same *position* value. This is why line 54 to 56 is needed. Suppose that there are 10 stems before the first appearance of the term phrase 'integer fact algorithm'. Some of the candidate phrases from this term phrase (i.e. integer, integer fact, and integer fact algorithm) have the same *position* value (i.e. 10).

3.3.6 Expanding Terms

Step 5 involves expanding single terms to term phrases. For each term, find all the term phrases that contain the term, and link it with the highest scoring term phrase. The result is a list of term phrases ordered by the scores calculated in Step 2.

This step ensures that no term links to more than one term phrase and uses the scores calculated in Step 2 to rank the output list of this step. The scores calculated in Step 4, on the other hand, may be discarded after this step. They are only useful for expanding single terms to term phrases, but are not useful for ranking the output list.

Suppose that the term phrase 'integer fact algorithm' and 'fact' appear in the first and second position of the term phrase set respectively. The term 'fact' (stem of 'factorization') will link to 'integer fact algorithm' instead of 'fact' (see Figure 3.3).

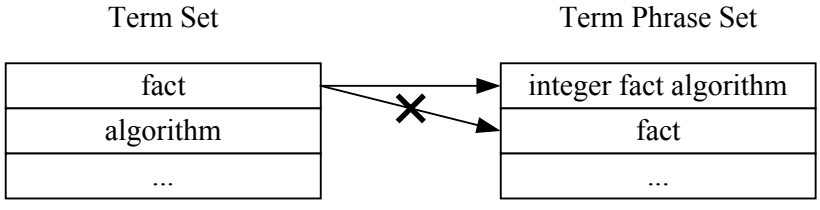

Figure 3.3: Expanding terms to term phrases

Here is the pseudocode for this step:

```
Step:      Expand Terms
Input:     List of terms, lt, and list of term phrases, ltp
Output:    List of term phrases, lltp, ordered by the scores
           calculated in Step 2
Method:
1          FOR each term t in lt
2           FOR each term phrase tp in ltp
3            IF tp contains t
4             Add tp to lctp
5            END IF
6           END FOR
7           Add the highest scoring term phrase in lctp to lltp
8          END FOR
```

3.3.7 Dropping Duplicates

Step 6 involves the elimination of duplicates from the list of term phrases. More than one term may link to the same term phrase (i.e. there may be converging arrows in the graph). If that is the case, the term phrase will be linked to the highest scoring term.

If there is more than one term linking to the same term phrase, that term phrase will appear more than once in the list of term phrases. If that is the case, we can simply keep the first appearance of that term phrase (because it is linked to the highest scoring term) and remove the rest from the list.

Suppose that the term ‘fact’ (appears in the first position of the term set) and ‘algorithm’ (appears in the second position) are expanded to the term phrase ‘integer fact algorithm’. ‘Fact’ instead of ‘algorithm’ will link to ‘integer fact algorithm’ (see Figure 3.4).

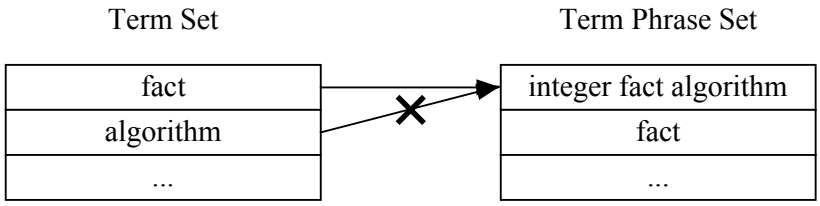

Figure 3.4: Deleting duplicate term phrases

Here is the pseudocode for this step:

```
Step:      Drop Duplicate
Input:     List of term phrases, lltp
Output:    List of term phrases, lutp
Method:
1          FOR each term phrase tp in lltp
2           IF tp is not in lutp
3            Add tp to lutp
4           END IF
5          END FOR
```

3.3.8 Displaying Output

Step 7 involves two closely related tasks:

1. Identify the most frequent corresponding phrase in the input document for each of the term phrases. If a term phrase is linked to more than one phrase, the most frequent phrase will be chosen.

2. Delete subphrases if they do not perform better than their superphrases. If phrase P_1 occurs within phrase P_2, P_1 is a subphrase of P_2 and P_2 is a superphrase of P_1. If a phrase is a subphrase of another phrase, it will only be accepted as a keyphrase if it is ranked higher; otherwise, it will be deleted from the output list.

Suppose that the term phrase ‘mathemat’ is linked to ‘mathematics’ (appears twice in the input document) and ‘mathematician’ (appears once). ‘Mathematics’ instead of ‘mathematician’ will be chosen for output (see Figure 3.5).

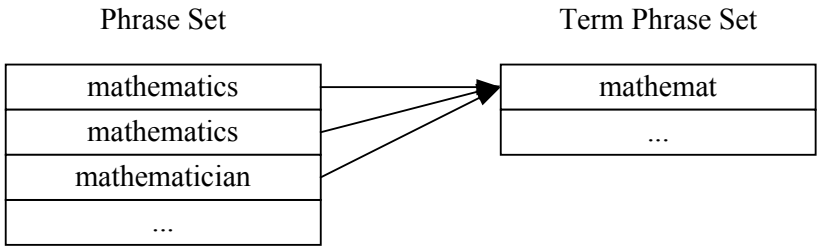

Figure 3.5: Identifying the most frequent phrases

Preliminary results suggest that the performance of KE can be boosted by performing some sort of post-processing on the output list before it is shown to the user. Some keyphrases could be redundant; similar keyphrases diminish the value of the summary generated. Therefore, Task 2 has been incorporated into our algorithm.

Suppose that the phrase ‘integer factorization algorithm’ appears in the first position of the output list and ‘factorization’ appears in the second position, ‘factorization’ will be removed because it is a subphrase of ‘integer factorization algorithm’ and is ranked lower (see Figure 3.6).

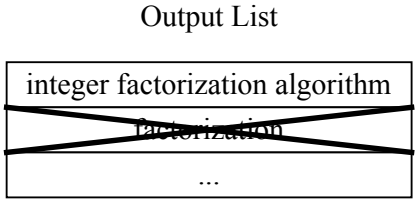

Figure 3.6: Deleting inferior subphrases

Here is the pseudocode for this step:

```
Step:      Display Output
Input:     List of term phrases, lutp
Output:    List of keyphrases, lk
Method:
```

```
1          // Get the most frequent corresponding phrase
2          FOR each term phrase tp in lutp
3
4           // Get a list of corresponding phrases
5           FOR each phrase p in lp
6            IF the stem of p is the same as tp
7             Add p to lcp
8            END IF
9           END FOR
10
11          // Get a list of unique phrases
12          FOR each phrase p in lcp
13           IF p is not in lup
14            Add p to lup
15           END IF
16          END FOR
17
18          // Get phrase frequency
19          FOR each phrase p in lup
20           Set PF to the number of occurrences of p in the input
           document
21          END FOR
22          Add the phrase with the highest PF to lk
23
24         END FOR
25
26         // Delete subphrases
27         FOR each phrase p in lk
28          FOR each phrase q in the part of lk that follows p
29           IF p contains q
30            Remove q from lk
31           END IF
32          END FOR
33         END FOR
```

The code from line 1 to 24 deals with Task 1, and line 26 to 33 deals with Task 2. The 'END FOR' in line 24 corresponds to the 'FOR' in line 2.

3.4 Training of KE

Keyphrase extraction is a classification problem: a document could be seen as a set of phrases, and a keyphrase extraction algorithm should correctly classify a phrase as a keyphrase or a non-keyphrase. We approach this problem from the perspective of machine learning, and treat it as a supervised learning task [29]. A collection of documents is used in our experiments. It is divided into two sets: training documents and testing documents. These two sets are disjoint. For details of these sets of documents, see Section 4.2. The training documents are used to tune KE. Each document consists of a set of keyphrase and non-keyphrase examples, and each example consists of a set of input values and the corresponding target output value. The testing documents are used to evaluate the generalization performance of the tuned KE.

The set of terms (i.e. output of Step 2) and the set of term phrases (i.e. output of Step 4) were tuned separately by a back-propagation neural network. The resulting sets were then combined to perform Step 5, 6 and 7 of the KE algorithm. After training, KE can be used to extract keyphrases from new documents. For details of the training results of KE, see Section 4.4.1. Neural networks are used because they are good at recognizing patterns and we do not have to understand how the input values are mapped to the target output values due to the networks' black box feature [6, 83, 86]. They have also proved useful in some IR projects [48, 113, 119]. For details of neural networks, see Section 2.8. Standard back-propagation techniques have been used to tune KE. We use the sigmoid function (or logistic function) as the activation function and the total squared error to calculate the associated error. We also use the momentum term, which is a common variation of the back-propagation algorithm, to speed up the training process. For details of these techniques, see [13, 29, 86].

To minimize the risk of overfitting, we use the cross-validation method [43] to estimate the appropriate point to stop training. Rumelhart *et al.* [86] argue that this method is reasonably powerful and simple, and often gives good results. It also has the advantage of drastically reducing the training time [25]. The set of training documents for KE is further divided into two sets: one used for training and the other used for cross-validation. These two sets are disjoint. For details of these sets of documents, see Section 4.2. Weights in the neural network are adjusted using the training set, but the error is computed using the validation set. In other words, the performance of the network is evaluated on the validation set. As long as the error for the validation set decreases (i.e. the network continues to improve on the validation set), the training continues. When the error begins to increase (i.e. the network begins to show poorer performance on the validation set), which is the point where overfitting occurs, the training stops. In short, overfitting is avoided in KE by stopping the training when the minimum of the validation set error is reached [29, 38, 72, 86].

The term set and the term phrase set are implemented as a vector of terms and a vector of term phrases during the implementation of KE. When KE is trained, a term phrase vector (or term vector) is used as an input vector of the neural network. Table 3.4 shows several elements in a term phrase vector representing a training document. The *TF*×*IDF* values are negative because Kea's IDF is used. For details of Kea's IDF, see Section 2.2.2. The reason why some *TF*×*IDF* values are zero is that these term phrases only occur in the document which contains them and do not occur in any other documents in the training corpus (i.e. IDF is 0).

Table 3.4: Elements in an input vector

Phrase	utility scores	services	conjoint analysis
Term Phrase	util scor	serv	conjoint an
TF×IDF	-0.000	-89.126	-0.000
Position	0.179	0.001	0.000
Title	0.000	0.000	1.000
Number of Terms	2.000	1.000	2.000

We also need another vector which contains an attribute indicating if a term phrase (or a term) is a keyphrase, and use this vector as a target output vector during the training of KE. If a term phrase is a keyphrase (or if a term occurs in a keyphrase), the attribute is set to 1; otherwise, it is set to 0. Table 3.5 shows a target output vector which corresponds to the input vector in Table 3.4.

Table 3.5: Elements in a target output vector

Phrase	utility scores	services	conjoint analysis
Term Phrase	util scor	serv	conjoint an
Flag for Keyphrase	1.000	0.000	1.000

Each training document consists of a set of keyphrase and non-keyphrase examples, and each example consists of a set of input values and the corresponding target output value. In KE, each training example consists of four input values and one target output value. Table 3.6 shows some examples used to train KE.

Table 3.6: Training examples

Phrase	utility scores	services	conjoint analysis
Term Phrase	util scor	serv	conjoint an
TF×IDF	-0.000	-89.126	-0.000
Position	0.179	0.001	0.000
Title	0.000	0.000	1.000
Number of Terms	2.000	1.000	2.000
Flag for Keyphrase	1.000	0.000	1.000

Different attributes in KE are measured on different scales (see Table 3.6). If they are directly used by the network, the effect of some attributes (e.g. *title*) might be completely dwarfed by attributes which have larger scales of measurement (e.g. *TF×IDF*). Therefore, it is common to normalize all the attribute values to lie in a fixed range, such as from 0 to 1. Normalization is particularly useful for classification algorithms involving neural networks. If the neural network back-propagation algorithm is used, normalizing the input values of each attribute will help to speed up the learning process [40].

If the maximum and minimum values are known, we could use the linear scaling function (or min-max normalization) to squash the input value into the 0-1 range. This function is used in KE because of its simplicity. The normalized value of the attribute *a* is calculated by [81, 114]:

$$\text{Normalized } a = \frac{a - min}{max - min} \tag{3.1}$$

where *max* and *min* are the maximum and minimum values in the vector that contains *a* respectively.

The linear scaling function works well if the maximum and minimum values are known. The range of the sample is always known, but the maximum and minimum values of the population may not be known sometimes. If the linear scaling function is used, values outside the sample range will be transformed into numbers that fall outside the 0-1 range. To solve this problem, the logistic function could be used [81]. The normalized value of the attribute *a* is given by:

$$\text{Normalized } a = \frac{1}{1+e^{-a}} \tag{3.2}$$

All the five attributes used in KE have been normalized by the linear scaling function. We have also evaluated other attributes in our experiments (see Section 4.4.2 for details). Some of them (such as *document length*) have been normalized by the logistic function. This is because the input document could be of any length, i.e. the maximum and minimum values of this attribute are unknown, and therefore the normalized value could fall outside the 0-1 range (if the linear scaling function is used).

3.5 Comparison with GenEx and Kea

KE is based on GenEx and Kea, but differs from them in several ways. For details of GenEx and Kea, see Section 2.2.1 and Section 2.2.2 respectively.

- Purely statistical methods are used in GenEx and Kea. KE, however, uses a combination of statistical and text processing techniques for keyphrase extraction. Part-of-speech tagging has been used to improve the quality of candidate phrases. Better quality data (i.e. better quality candidate phrases) often lead to better performance results [40, 114]. Only adjectives, verbs, nouns and noun phrases are selected as candidate phrases.

- KE uses a different set of attributes to discriminate between keyphrases and non-keyphrases: *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. Kea uses only two attributes: TF×IDF and distance. GenEx, on the other hand, uses many more attributes, but it does not use *TF×IDF* and *title*.

- KE uses a different machine learning algorithm; it is tuned by a neural network. GenEx is tuned by a genetic algorithm, while Kea is based on the naïve Bayes learning technique.

- KE is a different model; it consists of seven steps, and takes both words and phrases as candidate phrases. Kea is a simple model; it only selects phrases as candidate phrases, so it does not involve any linking between words and phrases. GenEx is more complicated; it consists of ten steps, considers both words and phrases, and involves many post-processing tasks.

The experimental results summarized in Table 4.17 (see Section 4.4.5) suggest that these differences make KE a better algorithm than either GexEx or Kea.

3.6 Summary

This chapter describes the KE algorithm. The algorithm has been described using pseudocode and explained with examples and informal English descriptions. In addition, we have introduced the training of KE and the normalization of different attributes evaluated in our experiments. The differences between KE, GenEx and Kea have also been discussed. The next chapter will present the experimental results relevant to KE.

CHAPTER FOUR

Extraction of Keyphrases from English Documents

4.1 Overview

This chapter presents the experimental results relevant to the KE algorithm. Testing has been carried out to validate and evaluate KE. KE has been tested on two different corpora. The first corpus is the same as the one used in GenEx and Kea, and it has been used to train and test KE in all our experiments (except the one discussed in Section 4.4.7). The criteria used for evaluating the output keyphrases are also the same as in GenEx and Kea, so direct comparison is possible. The second corpus is different and larger than the first one, and it has been used to test the generalization performance of KE. The evaluation criteria used for this corpus are the same as for the first corpus. We have evaluated the individual performance of different attributes and the performance of different combinations of attributes. The experiments suggest that *position* gives the best individual performance and that the best combination of attributes involves *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. In addition, we have compared different combinations of *TF×IDF*, and found that the standard TF and Kea's IDF gives the best performance. The experiments also indicate that KE performs better than other keyphrase extraction tools, including GenEx and Kea, and that it significantly outperforms Microsoft Word 2000's AutoSummarize feature. We have tried using the C4.5 decision tree learning method to tune KE, but the experiments show that neural networks are better for keyphrase extraction than this method. The domain independence of KE has also been confirmed in our experiments using the second corpus.

Section 4.2 introduces the corpus used to train and test KE. The criteria used for evaluating the output keyphrases are discussed in Section 4.3. Section 4.4 compares the individual performance of different attributes, the performance of different combinations of attributes and *TF*×*IDF*, the performance of different keyphrase extraction tools, and the performance of KE on different learning methods and different corpora. Section 4.5 discusses the performance results. Section 4.6 concludes this chapter.

4.2 Main Corpus

Turney [106, 107] uses five different corpora to train and test the GenEx algorithm (for details of GenEx, see Section 2.2.1). The five corpora are Journal Articles, Email Messages, Aliweb Web Pages, NASA Web Pages, and FIPS Web Pages. The links to these corpora (except for Email Messages) are provided in [107]. Frank *et al.* [34] use almost the same corpora to train and test the Kea algorithm (for details of Kea, see Section 2.2.2). Email Messages was the only corpus they could not access because it contained confidential messages. However, because of the ephemeral nature of the Web, most of these corpora are no longer available. Journal Articles is the only corpus we have access to (http://www.apperceptual.com/downloads/journals.zip).

A number of recent keyphrase extraction algorithms have been discussed in Section 2.2. However, these algorithms use different corpora (from the one used in GenEx and Kea) and the aim of some of these algorithms is different from ours, so we did not use those corpora in our experiments (for details, see Section 2.2). In addition, KE is based on GenEx and Kea, so it is better to use the same corpus (as the one used in these algorithms) so that our experimental results can be directly compared with theirs. We therefore use the Journal Articles corpus to train and test the KE algorithm. Because of the nature of this corpus, we believe it is possible to improve the performance of KE by removing common words used in academic writing (e.g. chapter, paper, etc) and using indicator phrases to ignore words that occur in specific sections (e.g. References). However, we decided not to do this because we want KE to be domain-independent.

The Journal Articles corpus contains 75 articles selected from five different journals. Three of these journals are about cognition, one is about hotel industry, and one is about chemistry. Please see Table 4.1 for the sources of the Journal Articles corpus. All these articles contain keyphrases supplied by the authors.

Table 4.1: Sources of the Main Corpus

Journal Name	Field	Number of Documents
Psycoloquy	Cognition	20
The Neuroscientist	Cognition	2
Behavioural and Brain Sciences Preprint Achieve	Cognition	33
Journal of the International Academy of Hospitality Research	Hotel industry	6
Journal of Computer-Aided Molecular Design	Chemistry	14

Psycoloquy has been used to test KE and the remaining journals have been used for training and cross-validation. Psycoloquy makes up about 25% of the corpus and the other journals make up the remaining 75%. Also, we would like to test the ability of KE to generalize across journals (i.e. domain independence), so it is best if the testing set does not contain any journal article from the training set[10].

Table 4.2 shows that most of the documents in the training set are much longer than those in the testing set.

Table 4.2: Number of words per document (Main Corpus)

Corpus	Average	Range	Standard Deviation
Training	13120.00	3150.00-33575.00	7747.47
Testing	4350.20	1010.00-12449.00	2725.50

[10] Five documents have been randomly selected from the training set to form the validation set.

Table 4.3 shows that most of the documents contain seven to eight keyphrases.

Table 4.3: Number of keyphrases per document (Main Corpus)

Corpus	Average	Range	Standard Deviation
Training	7.13	3.00-12.00	2.58
Testing	8.35	4.00-17.00	3.12

Table 4.4 shows that most of the keyphrases contain one to two words. Keyphrases often contain only characters and rarely contain numbers. Only two out of 75 documents provide keyphrases containing numbers. Both of these documents are concerned with 3-D (three dimensions).

Table 4.4: Number of words per keyphrase (Main Corpus)

Corpus	Average	Range	Standard Deviation
Training	1.75	1.13-2.60	0.37
Testing	1.53	1.08-2.43	0.31

Table 4.5 shows that about 80% of the keyphrases can be found in the document. Keyphrases and documents have been stemmed before they are compared (for details, see Section 4.3).

Table 4.5: Percentage of keyphrases found in the document (Main Corpus)

Corpus	Average	Range	Standard Deviation
Training	0.84	0.44-1.00	0.17
Testing	0.78	0.50-1.00	0.16

Table 4.6 confirms that titles are a good source of high quality keyphrases.

Table 4.6: Number of keyphrases found in the title (Main Corpus)

Corpus	Average	Range	Standard Deviation
Training	1.00	0.00-3.00	0.98
Testing	1.00	0.00-3.00	0.97

4.3 Evaluation

This section discusses the criteria, the stemming algorithm, and the performance measures used for assessing the quality of the output keyphrases.

4.3.1 Criteria

We need some ways to assess the ability of KE to extract keyphrases and reject non-keyphrases. Asking human assessors to evaluate machine-extracted keyphrases seems the most direct way, but this is costly. It is more common to compare machine-extracted keyphrases with author-assigned keyphrases. Jones and Paynter [50, 51] argue that authors do provide good quality keyphrases, so it is reasonable to use them as a standard of comparison for evaluating machine-extracted keyphrases. They also suggest that author-assigned keyphrases are listed with the most important keyphrases first, which might have some implications when author-assigned keyphrases are used to measure keyphrase quality.

If the author suggests that 'concepts' is a keyphrase and the computer provides 'concept' as an output keyphrase, they should be considered the same. On the other hand, 'thinking' should be considered different from 'analogical thinking', and 'hominid evolution' should be considered different from 'evolution'. A machine-extracted keyphrase is said to be correct if its stem matches the stem of an author-assigned keyphrase. Abbreviations are therefore considered different from their complete form, e.g. 'EEG' is not the same as 'electroencephalogram'. The word order in keyphrases is also important, e.g. 'data archiving' is different from 'archiving data'. However, keyphrases with and without a hyphen (-) or a slash (/) should be considered the same, e.g. 'tactile-kinesthetic body' is the same as 'tactile kinesthetic body'. Table 4.7 shows an example of the keyphrases extracted by KE. Keyphrases which match their corresponding author-assigned keyphrases according to the iterated Lovins stemmer are said to be correct and are in bold type.

Table 4.7: Examples of correct and incorrect keyphrases

Title	Precis of: The Roots of Thinking
Author-assigned Keyphrases	Analogical thinking, animate form, concepts, evolution, tactile-kinesthetic body
Machine-extracted Keyphrases (Top 5)	Thinking, **concept**, **tactile kinesthetic body**, hominid evolution, thesis

4.3.2 Stemming

Stemming plays an important role in determining if a machine-extracted keyphrase is correct. Many stemming algorithms have been proposed. The Porter and the Lovins algorithms are the two most common ones developed for English documents [46, 56]. Further details of the Porter and the Lovins algorithms can be found in Section 2.6. Table 4.8 shows some examples of the behaviour of different stemming algorithms. These examples are from [108]. The Lovins stemmer correctly recognizes that 'science' and 'scientist' have the same stem, but the Porter considers them different. On the other hand, the Lovins stemmer incorrectly maps 'police' and 'policy' to the same stem (i.e. over-stemming error), but the Porter correctly recognizes them as different words.

The iterated Lovins stemmer has been used to evaluate the performance of KE. Further details of this stemmer can be found in Section 2.6.3. The iterated Lovins stemmer correctly recognizes that 'science' and 'scientist' have the same stem, but incorrectly maps 'police' and 'policy' to 'pol' (see Table 4.8). It also fails to recognize that 'assemblies' and 'assembly' have the same stem (i.e. under-stemming error) despite the fact that it is the most aggressive stemming algorithm among the three.

Table 4.8: Examples of the behaviour of different stemming algorithms [108]

Word	Porter Stemmer	Lovins Stemmer	Iterated Lovins Stemmer
Science	Scienc	Sci	Sc
Scientist	Scientist	Sci	Sc
Police	Polic	Polic	Pol
Policy	Polici	Polic	Pol
Assemblies	Assembli	Assembl	Assembl
Assembly	Assembli	Assemb	Assemb

4.3.3 Recall and Precision

As mentioned in Section 1.3.12, recall and precision are the two most common measures of retrieval performance. Recall is the proportion of the relevant documents that are retrieved in a search, while precision is the proportion of the documents retrieved in a search that are relevant [88, 95]. It makes sense to define recall and precision using terms such as 'relevant', 'documents' and 'retrieved' because information retrieval is about the retrieval of relevant documents. These terms, however, are not applicable to the problem of keyphrase extraction because keyphrase extraction is about the extraction of correct keyphrases. Therefore, we need to adapt the standard definition of recall and precision.

Keyphrase extraction is a classification task. A phrase can be classified as a keyphrase or a non-keyphrase by the author or the machine. The four possible cases could be illustrated by a matrix [107] (see Table 4.9).

Table 4.9: Matrix for keyphrase classification

	Classified as a keyphrase by the author	Classified as a non-keyphrase by the author
Classified as a keyphrase by the machine	A	B
Classified as a non-keyphrase by the machine	C	D

When recall and precision are used to evaluate keyphrase extraction algorithms, recall is the proportion of correct keyphrases extracted, while precision is the proportion of extracted keyphrases that are correct. For the problem of keyphrase extraction, recall and precision are calculated by [107]:

$$\text{Recall} = \frac{A}{A+C} \tag{4.1}$$

$$\text{Precision} = \frac{A}{A+B} \tag{4.2}$$

4.4 Experimental Results

This section provides the training results, and compares the individual performance of different attributes, the performance of different combinations of attributes and $TF \times IDF$, the performance of different keyphrase extraction tools, and the performance of KE on different learning methods and different corpora.

4.4.1 Training

KE has to be trained before it can be applied to new documents for keyphrase extraction. The set of terms (i.e. output of Step 2) and the set of term phrases (i.e. output of Step 4) were tuned separately by a fully connected 4-9-1 back-propagation neural network (for details of neural networks, see Section 2.8). The resulting sets were then combined to perform Step 5, 6 and 7 of the KE algorithm (for details of KE, see Section 3.2). The number of input units and output units of a neural network are constrained by training examples. Since each training example in KE consists of four input values and one target output value (see Section 3.4), there are four input units and one output unit. The number of hidden units and hidden layers, however, are variable. The number of hidden units affects the generalization performance of a neural network. We have tested different numbers of hidden units, and found that nine hidden units give the best result. Also, it is possible to have more than one hidden layer in a neural network, but one hidden layer is adequate for most applications. KE has been tuned and tested on a neural network with two hidden layers, but the experiments indicate that the difference between that and one hidden layer is small. Therefore, only one hidden layer is used.

During training, the initial weights of the neural network used to tune KE were set to random values ranged from –0.5 to +0.5. We have also tried different values of the learning rate and the momentum term, and found that the network gives the best performance when the learning rate is set to 0.1 and the momentum term to 0.5.

The experiments also indicate that the term set often requires more training iterations than the term phrase set. A training iteration (or epoch) involves all the documents in the training set and the selection of 150 term phrases (and terms), including both keyphrase and non-keyphrase examples (and keyword and non-keyword examples), from each document. During training, the author-assigned keyphrases are used as keyphrase examples, and phrases (other than those supplied by the author as keyphrases) are randomly selected from the document as non-keyphrase examples. The term set works in a similar way, except that the author-assigned keyphrases are tokenized (using white spaces), and the resulting words are used as keyword examples, and words (other than those in the author-assigned keyphrases) are randomly selected as non-keyword examples. Please refer to Section 3.4 for further information about when to stop training.

4.4.2 Different Attributes

We have evaluated a number of attributes in our experiments: six of them have been discussed in Section 1.3, and the remaining attributes will be discussed in this section (these attributes are not introduced earlier because they are used only in this chapter and, as we will see later in this section, the experiments show that they are not useful for keyphrase extraction).

Six attributes have been discussed so far: *TF×IDF*, *position*, *title*, *proper noun*, *number of terms*, and *document length*. For details of these attributes, see Section 1.3. In addition to these attributes, seven attributes (i.e. *inverse term frequency*, *number of characters*, *average paragraph length*, *average paragraph position*, *average sentence length*, *average sentence position*, and *topic sentence*) have been evaluated in our experiments. Their definitions are as follows:

- The attribute *inverse term frequency* (ITF) is the rarity of a term across the collection. It is similar to IDF, but is defined differently. The *ITF* of a term *T* in a document *D* is given by:

$$\text{ITF} = \frac{\text{no. of occurrences of } T \text{ in } D}{\text{no. of occurrences of } T \text{ throughout collection}} \tag{4.3}$$

- The attribute *number of characters* is the number of characters in a term.

- The attribute *average paragraph length* (APL) is the average number of words in the paragraphs that contain a term. The *APL* of a term *T* in a document *D* is calculated by:

$$\text{APL} = \frac{\text{no. of words in paragraphs that contain } T}{\text{no. of occurrences of } T \text{ in } D} \tag{4.4}$$

- The attribute *average paragraph position* (APP) is the average position of the paragraphs that contain a term. The *APP* of a term *T* in a document *D* is given by:

$$\text{APP} = \frac{\text{position of paragraphs that contain } T}{\text{no. of occurrences of } T \text{ in } D} \tag{4.5}$$

- The attribute *average sentence length* (ASL) is the average number of words in the sentences that contain a term. The *ASL* of a term *T* in a document *D* is calculated by:

$$\text{ASL} = \frac{\text{no. of words in sentences that contain } T}{\text{no. of occurrences of } T \text{ in } D} \tag{4.6}$$

- The attribute *average sentence position* (ASP) is the average position of the sentences that contain a term. The *ASP* of a term *T* in a document *D* is given by:

$$\text{ASP} = \frac{\text{position of sentences that contain } T}{\text{no. of occurrences of } T \text{ in } D} \tag{4.7}$$

- The attribute *topic sentence* is a flag that indicates if a term occurs in the topic sentences of a document. A topic sentence is usually the first sentence of a paragraph and is intended to give readers an idea of what the paragraph is going to be about. If a term occurs in the first sentence of a paragraph, *topic sentence* is set to 1; otherwise, it is set to 0.

We have compared the individual performance of different attributes. Two attributes (i.e. *number of terms* and *document length*) have not been evaluated in this experiment. Recall that each element of the term set is associated with at most one element of the term phrase set (for details, see Section 3.3.6). Since *number of terms* is always one when it comes to single terms, the attribute (if used alone) cannot discriminate between different terms. Therefore, we decided not to evaluate the individual performance of this attribute. Similarly, *document length* is always the same when it comes to terms and term phrases that are from the same document. Since this attribute cannot discriminate between those terms and term phrases, we did not compare its individual performance with that of other attributes.

Attributes are divided into three groups according to their individual performance results. Group A consists of the attributes that give the best performance: *TF*×*IDF*, *position*, and *title*. Group B consists of *proper noun*, *ITF*, and *number of characters*. Group C consists of the attributes that give the worst performance: *APL*, *APP*, *ASL*, *ASP*, and *topic sentence*.

Table 4.10 shows the individual performance of *TF×IDF* (using the standard TF and Kea's IDF), *position*, and *title*. The experiments indicate that the performance of *position* is more stable than that of *TF×IDF*. The average precision of *position* lies between 0.21 and 0.25, while that of *TF×IDF* lies between 0.16 and 0.35. In addition, there is a tendency for the average precision of *TF×IDF* to fall. The experiments also show that the performance of *position* is always better than that of *title*. We conclude that *position* is the best individual indicator of keyphrase extraction. This confirms the findings by Edmundson [28] and Kupiec *et al.* [58] that location-based methods give the best performance, though their work is concerned with sentence extraction and they use a different set of attributes. For details of their work, see Section 2.5. Figure 4.1 shows the comparison of the individual performance of different attributes in Group A with varying number of output keyphrases.

Table 4.10: Individual performance of different attributes (Group A)

Attribute	Number of Keyphrases	Average Number of Correct Keyphrases	Standard Deviation
TF×IDF	1	0.35	0.49
	2	0.55	0.69
	3	0.70	0.92
	4	0.80	1.06
	5	0.80	1.11
Position	1	0.25	0.44
	2	0.45	0.76
	3	0.65	0.88
	4	0.90	0.97
	5	1.05	1.10
Title	1	0.25	0.44
	2	0.35	0.59
	3	0.60	0.82
	4	0.80	1.01
	5	0.90	1.17

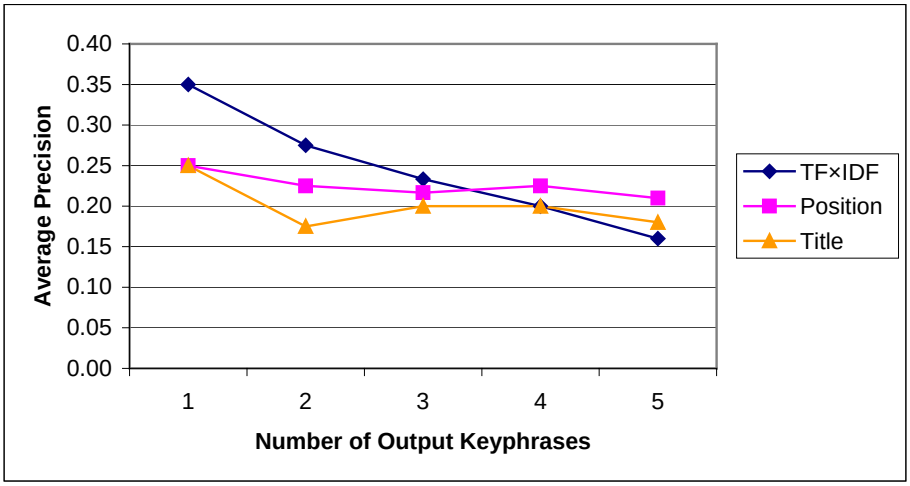

Figure 4.1: Comparison of the individual performance of different attributes (Group A)

Table 4.11 shows the individual performance of *proper noun*, *ITF*, and *number of characters*. The experiments suggest that these attributes give similar performance results when the desired number of output keyphrases is set to three or above. When the number of output keyphrases is set to three, four and five, the differences between the average precision of these attributes are 0.00, 0.01 and 0.02 respectively. The individual performance of *proper noun* is worse than we expected. We believe this is because of the nature of the corpus used (i.e. a collection of journal articles) and the fact that *proper noun* is not a useful indicator on its own. The References section of a journal article always occurs at the end of the document and contains many proper nouns (e.g. author names). These proper nouns have a negative effect on those useful ones that appear in the body of the document. They dwarf those useful proper nouns and reduce their chance of being output. This is why *proper noun* is not a useful indicator on its own. However, as we will see in Section 4.4.3, *proper noun* is more useful when it is combined with attributes such as *position*. This is because *position* gives information about the location of a proper noun in the document: terms with small/ medium *position* and *proper noun* set to one (i.e. proper nouns that appear in the body of the document) are preferred to terms with large *position* and *proper noun* set to one (i.e. proper nouns that appear in the References section). Figure 4.2 shows the comparison of the individual performance of different attributes in Group B with varying number of output keyphrases.

Table 4.12 shows the individual performance of *APL*, *APP*, *ASL*, *ASP*, and *topic sentence*. The experiments suggest that these attributes are not useful for extracting keyphrases from documents. In fact, *ASL* fails to identify any keyphrase. This means that sentence length is not relevant to keyphrase extraction. The number of words in a paragraph, however, is more useful than that in a sentence. We believe this is because authors often devote some space to discussing keyphrases in their documents, so it is unlikely that they occur in very short paragraphs. The experiments also show that the performance of *APL* is the same as that of *topic sentence*, and that the performance of *APP* is the same as that of *ASP*. Figure 4.3 shows the comparison of the individual performance of different attributes in Group C with varying number of output keyphrases.

Table 4.11: Individual performance of different attributes (Group B)

Attribute	Number of Keyphrases	Average Number of Correct Keyphrases	Standard Deviation
Proper Noun	1	0.10	0.31
	2	0.15	0.37
	3	0.20	0.41
	4	0.20	0.41
	5	0.20	0.41
ITF	1	0.05	0.22
	2	0.05	0.22
	3	0.20	0.41
	4	0.25	0.55
	5	0.30	0.57
No. of Characters	1	0.10	0.31
	2	0.10	0.31
	3	0.20	0.52
	4	0.25	0.55
	5	0.25	0.55

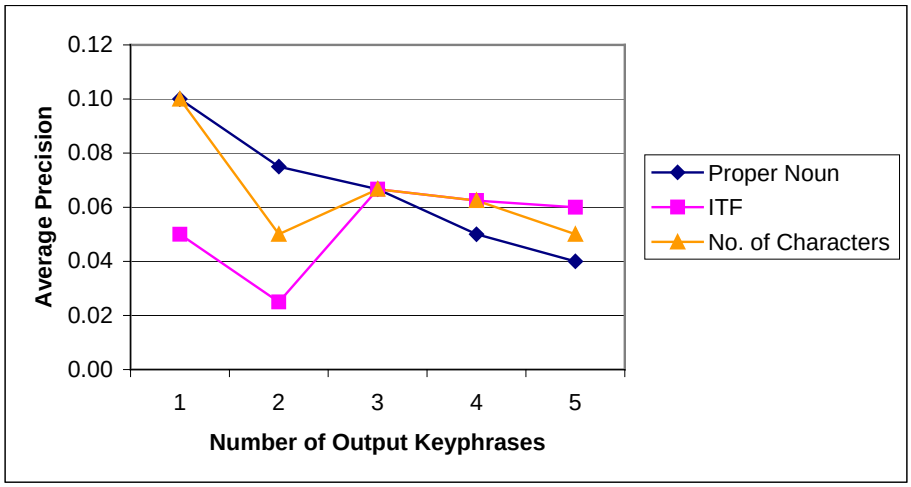

Figure 4.2: Comparison of the individual performance of different attributes (Group B)

Table 4.12: Individual performance of different attributes (Group C)

Attribute	Number of Keyphrases	Average Number of Correct Keyphrases	Standard Deviation
APL	1	0.00	0.00
	2	0.05	0.22
	3	0.10	0.31
	4	0.10	0.31
	5	0.10	0.31
APP	1	0.00	0.00
	2	0.00	0.00
	3	0.00	0.00
	4	0.05	0.22
	5	0.10	0.31
ASL	1	0.00	0.00
	2	0.00	0.00
	3	0.00	0.00
	4	0.00	0.00
	5	0.00	0.00
ASP	1	0.00	0.00
	2	0.00	0.00
	3	0.00	0.00
	4	0.05	0.22
	5	0.10	0.31
Topic Sentence	1	0.00	0.00
	2	0.05	0.22
	3	0.10	0.31
	4	0.10	0.31
	5	0.10	0.31

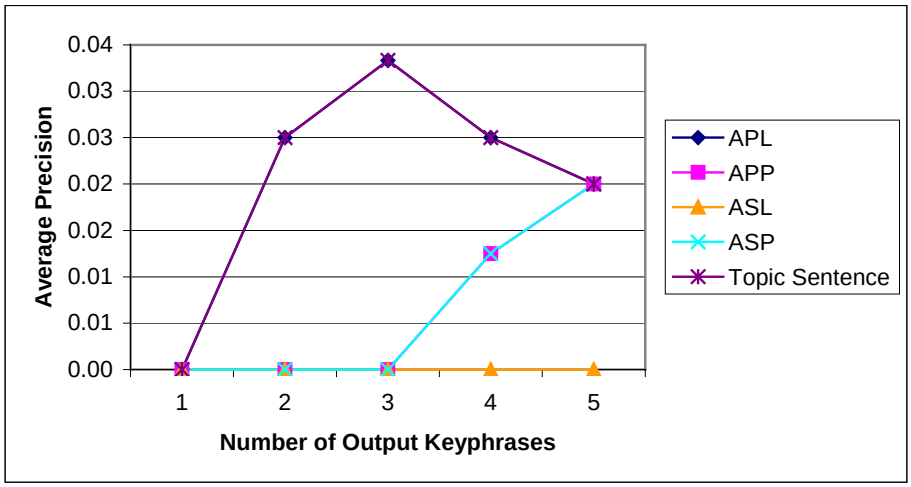

Figure 4.3: Comparison of the individual performance of different attributes (Group C)

4.4.3 Different Combinations of Attributes

We have compared the performance of different combinations of attributes. As shown in the experiment discussed in Section 4.4.2, *TF×IDF* (using the standard TF and Kea's IDF), *position* and *title* are useful for identifying high quality keyphrases, so we have combined these attributes to form the *basic model*. The experiment discussed in Section 4.4.2 also indicates that *APL*, *APP*, *ASL*, *ASP*, and *topic sentence* are not useful for keyphrase extraction, so they were excluded from this experiment. In other words, only attributes in Group A and Group B are used in this experiment.

A number of models have been built based on the basic model. They are divided into two groups according to the number of attributes added to the basic model. Group D consists of the models with only one attribute added to the basic model: *basic model + proper noun*, *basic model + ITF*, *basic model + number of characters*, *basic model + number of terms*, and *basic model + document length*. Group E consists of the models with two attributes added to the basic model: *basic model + proper noun + number of terms*, *basic model + ITF + number of terms*, *basic model + number of characters + number of terms*, and *basic model + document length + number of terms*.

Table 4.13: Performance of different combinations of attributes (Group D)

Attribute	Number of Keyphrases	Average Number of Correct Keyphrases	Standard Deviation
Basic Model	1	0.30	0.47
	2	0.45	0.69
	3	0.75	0.97
	4	0.95	1.32
	5	1.30	1.49
Basic Model + Proper Noun	1	0.30	0.47
	2	0.55	0.76
	3	0.80	1.01
	4	1.15	1.27
	5	1.30	1.49
Basic Model + ITF	1	0.35	0.49
	2	0.50	0.69
	3	0.85	0.93
	4	1.05	1.28
	5	1.25	1.48
Basic Model + No. of Characters	1	0.30	0.47
	2	0.55	0.76
	3	0.90	1.02
	4	1.15	1.35
	5	1.25	1.37
Basic Model + No. of Terms	1	0.30	0.47
	2	0.55	0.76
	3	0.80	1.01
	4	1.25	1.21
	5	1.35	1.23

Basic Model + Document Length	1	0.30	0.47
	2	0.55	0.76
	3	0.85	0.99
	4	1.15	1.27
	5	1.20	1.36

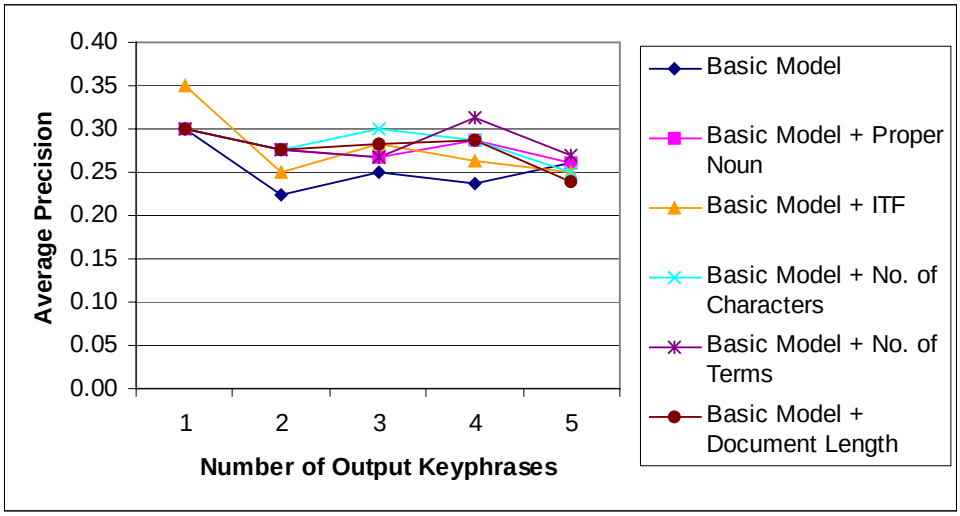

Figure 4.4: Comparison of different combinations of attributes (Group D)

Table 4.13 shows the performance of different models in Group D. The experiments show that all these models give similar performance (the average precision of these models lies mostly in the range of 0.25 to 0.30), though *basic model + number of terms* appears to give the best result. Figure 4.4 shows the comparison of different combinations of attributes in Group D with varying number of output keyphrases.

As shown in the previous experiment, *basic model + number of terms* gives the best result, so we have carried out more experiments by adding different attributes to this model.

Table 4.14 shows the performance of different models in Group E. The experiments show that *basic model + proper noun + number of terms* performs better than the other models, and that the differences between these models are small when the desired number of output keyphrases is set to three or below. When the number of output keyphrases is set to one, two and three, the differences between the average precision of these models are 0.00, 0.03 and 0.03 respectively. We conclude that the best combination consists of five attributes: *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. Figure 4.5 shows the comparison of different combinations of attributes in Group E with varying number of output keyphrases.

4.4.4 Different Combinations of TF×IDF

As mentioned in Section 2.3.3, there is no universal definition of *TF×IDF*. Four different *TF×IDF* definitions have been discussed in this thesis: standard TF, standard IDF, normalized TF, and Kea's IDF. Please refer to Section 1.3.5 for further information about the standard TF and the standard IDF, to Section 2.2.2 for Kea's IDF, and to Section 2.3.3 for the normalized TF. Three different combinations of *TF×IDF* have been implemented using these definitions and tested in our experiments.

Table 4.15 shows the performance of different *TF×IDF* combinations. The experiments show that the difference between the standard TF and Kea's IDF and the standard TF and standard IDF is small, though the former tends to give more stable results. The average precision of the standard TF and Kea's IDF lies between 0.30 and 0.34, while that of the standard TF and standard IDF lies between 0.27 and 0.35. The average precision of the normalized TF and standard IDF lies between 0.22 and 0.40, and has a tendency to fall. Figure 4.6 shows the comparison of different *TF×IDF* combinations with varying number of output keyphrases.

Table 4.14: Performance of different combinations of attributes (Group E)

Combination of Attributes	Number of Keyphrases	Average Number of Correct Keyphrases	Standard Deviation
Basic Model + Proper Noun + No. of Terms	1	0.30	0.47
	2	0.65	0.75
	3	1.00	0.97
	4	1.35	1.23
	5	1.50	1.32
Basic Model + ITF + No. of Terms	1	0.30	0.47
	2	0.60	0.68
	3	0.90	0.91
	4	1.25	1.16
	5	1.35	1.14
Basic Model + No. of Characters + No. of Terms	1	0.30	0.47
	2	0.65	0.75
	3	0.95	1.05
	4	1.10	1.25
	5	1.35	1.39
Basic Model + Document Length + No. of Terms	1	0.30	0.47
	2	0.60	0.75
	3	0.95	0.94
	4	1.15	1.14
	5	1.30	1.22

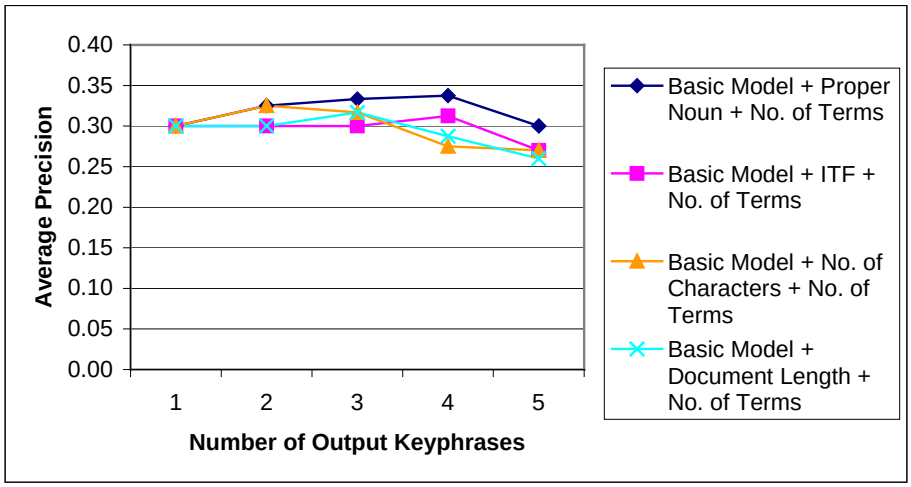

Figure 4.5: Comparison of different combinations of attributes (Group E)

Table 4.15: Performance of different combinations of TF×IDF

Combination of TF×IDF	Number of Keyphrases	Average Number of Correct Keyphrases	Standard Deviation
Standard TF and Kea's IDF	1	0.30	0.47
	2	0.65	0.75
	3	1.00	0.97
	4	1.35	1.23
	5	1.50	1.32
Standard TF and Standard IDF	1	0.35	0.49
	2	0.70	0.57
	3	0.85	0.75
	4	1.15	0.99
	5	1.35	1.09
Normalized TF and Standard IDF	1	0.40	0.50
	2	0.65	0.81
	3	0.85	1.04
	4	0.95	1.00
	5	1.10	0.97

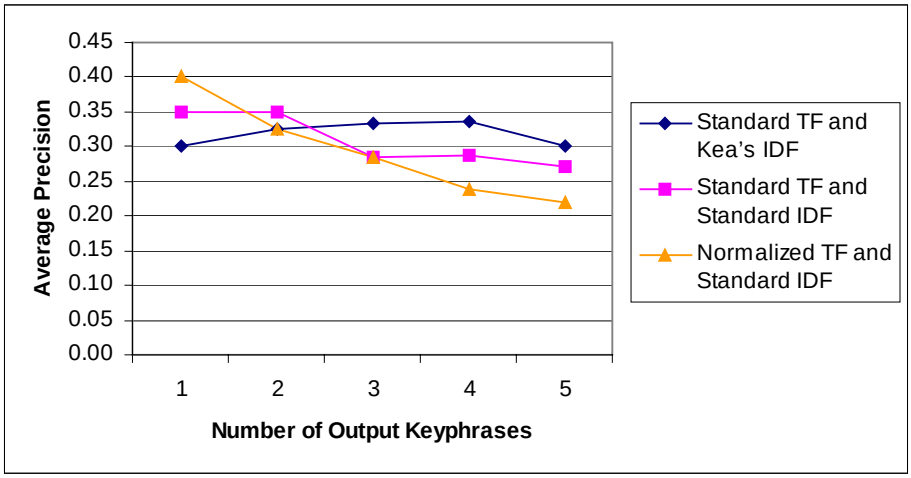

Figure 4.6: Comparison of different combinations of TF×IDF

4.4.5 Different Keyphrase Extraction Tools

We have compared the performance of KE with that of other keyphrase extraction tools: GenEx, C4.5[11], Kea, Kea-C4.5[12], and Microsoft Word 2000 (the *AutoSummarize*[13] feature). C4.5 and Kea-C4.5 have not been discussed in detail because they have mainly been used as a standard of comparison for evaluating the performance of GenEx and Kea respectively. Please refer to [34, 107] for further information about C4.5 and Kea-C4.5, and to Section 2.2 for GenEx and Kea. Microsoft Word was chosen because it was a very popular word processing tool with the extraction of keywords and key sentences feature. Five keyphrases have been extracted from each testing document by these tools and compared with the corresponding author-assigned keyphrases. The number of output keyphrases is set to five because AutoSummarize always generates exactly five keyphrases. Also, unlike the other tools, AutoSummarize cannot be trained and the output keyphrases always contain exactly one word. Table 4.16 shows the keywords extracted by AutoSummarize from a testing document. Correct keywords are in bold type.

Table 4.16: Example of the keywords extracted by AutoSummarize

Title	The Base Rate Fallacy Myth
Author-assigned Keyphrases	Base rate fallacy, Bayes' theorem, decision making, ecological validity, ethics, fallacy, judgement, probability
Keywords	Rate, base, information, **judgement**, psychology

[11] C4.5 consists of a set of parameterized heuristic rules that are fine-tuned by the C4.5 decision tree learning algorithm. Some of these parameters are used in GenEx.

[12] Kea-C4.5 is a variation of Kea. The pre- and post-processing are the same as in Kea. The only difference is that it uses the C4.5 decision tree learning algorithm, instead of the naïve Bayes learning algorithm.

[13] The AutoSummarize feature aims at extracting key sentences from a given document and is available from the *Tools* menu. The generation of keywords is actually a by-product of AutoSummarize. When AutoSummarize is used, it also fills in the *Keywords* field of the document's *Properties*, which is available from the *File* menu.

Table 4.17 shows the number of correct keyphrases identified by different keyphrase extraction tools. Direct comparison is possible because all the tools have been trained (except AutoSummarize as it cannot be trained) and tested on the same set of documents. Results of GenEx, C4.5, Kea, and Kea-C4.5 are from [34]. We tried to get more statistical data from the authors of these methods, but since the experiments were carried out in the late 1990s, the data are no longer available. The experiments indicate that KE (using the standard TF and Kea's IDF) performs better than the other methods (in terms of the average number of correct keyphrases). The data available for the other methods are insufficient for us to show that the difference between KE and any of these methods is statistically significant. However, in all cases, the mean for KE exceeds the mean for the other methods. Thus, the results are very suggestive, although not conclusive. The distribution of the performance results of KE is positively skewed, and we believe this is likely the case for the other methods. This may explain why the standard deviations of KE and the other methods are quite large. Since Word 2000 can only extract five single words from each document and most of the keyphrases in the corpus contain more than one word, it is not surprising that Word 2000 gives the worst performance.

Table 4.18 shows the keyphrases extracted by KE from three testing documents. Correct keyphrases are in bold type. KE extracts zero, five, and two correct keyphrases in the first, second and third example respectively. Nevertheless, if we look at the first example carefully, we will find that KE has actually extracted one correct keyphrase (i.e. 'cell assemblies'). This keyphrase is considered different from the author-assigned keyphrase 'cell assembly' because of the stemmer employed. This will be discussed in detail in Section 4.5.

4.4.6 Different Learning Methods

In addition to neural networks, we have tried using the C4.5 decision tree learning algorithm [82] to tune KE. There are two reasons for doing this:

- Different machine learning methods should give approximately the same performance results, but some methods might be more suitable for the problem of keyphrase extraction than others. As shown in the experiment discussed in Section 4.4.5, the choice of a learning method does affect the performance of a keyphrase extraction algorithm: GenEx and Kea give different results when they are tuned by different learning methods.

- The experiment discussed in Section 4.4.5 suggests that KE performs better than the other keyphrase extraction tools. Nevertheless, the improvement in performance could be a result of the algorithm (and the selection of attributes) itself and/ or the learning method (i.e. neural networks) employed. Both GenEx and Kea have been tuned by the C4.5 learning method, and the tuned algorithms have been used as a standard of comparison for evaluating the performance of GenEx and Kea. If KE is tuned by the C4.5 learning method, we can exclude the effect of neural networks and evaluate only the performance of the algorithm.

Table 4.17: Performance of different keyphrase extraction tools

	Average Number of Correct Keyphrases	Standard Deviation
KE	1.50	1.32
GenEx	1.45	1.24
C4.5	1.40	1.28
Kea	1.35	0.93
Kea-C4.5	1.20	0.83
Word 2000	0.85	0.93

Table 4.18: Examples of the keyphrases extracted by KE

Title	Brain Rhythms, Cell Assemblies and Cognition: Evidence from the Processing of Words and Pseudowords
Author-assigned Keyphrases	Brain theory, cell assembly, cognition, event related potentials, ERP, electroencephalograph, EEG, gamma band, Hebb, language, lexical processing, magnetoencephalography, MEG, psychophysiology, periodicity, power spectral analysis, synchrony
Machine-extracted Keyphrases (Top 5)	Words, processing, cell, cell assemblies, spatiotemporal activity patterns

Title	Precis of: Metapsychology: Missing Links in Behavior, Mind, and Science
Author-assigned Keyphrases	Behavior, causality, experimentation, explanation, introspection, mind-body problem, observation, philosophy, psychology, reductionism, science, theory
Machine-extracted Keyphrases (Top 5)	**Science**, **psychology**, **theory**, **explanation**, **behavior**

Title	Precis of: The Roots of Thinking
Author-assigned Keyphrases	Analogical thinking, animate form, concepts, evolution, tactile-kinesthetic body
Machine-extracted Keyphrases (Top 5)	Thinking, **concept**, **tactile kinesthetic body**, hominid evolution, thesis

The C4.5 learning algorithm is an unstable classification algorithm, i.e. the constructed classifier (in the form of a decision tree) is sensitive to small changes to the training data, so bagging has been used to improve performance by reducing variance [101, 114]. Both GenEx and Kea have been tuned by 50 bagged C4.5 decision trees [34]. To ensure comparability, the same has been carried out on KE. The set of terms (i.e. output of Step 2) and the set of term phrases (i.e. output of Step 4) were tuned separately by 50 bagged C4.5 decision trees. The resulting sets were then combined to perform Step 5, 6 and 7 of the KE algorithm (for details of KE, see Section 3.2).

There are a number of options, which allow users of the C4.5 program [82] to improve decision tree performance, such as the *–c* option and the *–m* option. The *–c* option sets the confidence threshold for pruning, and the *–m* option sets the minimum number of examples needed to form a leaf of the decision tree.

We have evaluated the performance of different numbers of training examples and different values of *–c* and *–m*, and found that KE gives the best performance when 200 terms and 150 term phrases are selected from each training document with *–c* set to 50% and *–m* to 10. The experiments also indicate that, in general, simple trees give better results than bushy trees. We believe this is because bushy trees tend to be overtrained on the training set.

Table 4.19 shows the performance of KE and *KE-C4.5* (i.e. KE tuned by the C4.5 learning method). The experiments indicate that the performance of KE is more stable than that of KE-C4.5 (the average precision of KE lies between 0.30 and 0.34, while that of KE-C4.5 lies between 0.27 and 0.35), and that KE often gives better performance results than KE-C4.5, except when the desired number of output keyphrases is set to one. We conclude that neural networks are better for keyphrase extraction than the C4.5 learning algorithm. Figure 4.7 shows the comparison of KE and KE-C4.5 with varying number of output keyphrases.

Table 4.19: Performance of KE and KE-C4.5

Algorithm	Number of Keyphrases	Average Number of Correct Keyphrases	Standard Deviation
KE	1	0.30	0.47
	2	0.65	0.75
	3	1.00	0.97
	4	1.35	1.23
	5	1.50	1.32
KE-C4.5	1	0.35	0.49
	2	0.55	0.60
	3	0.90	0.91
	4	1.15	1.23
	5	1.35	1.27

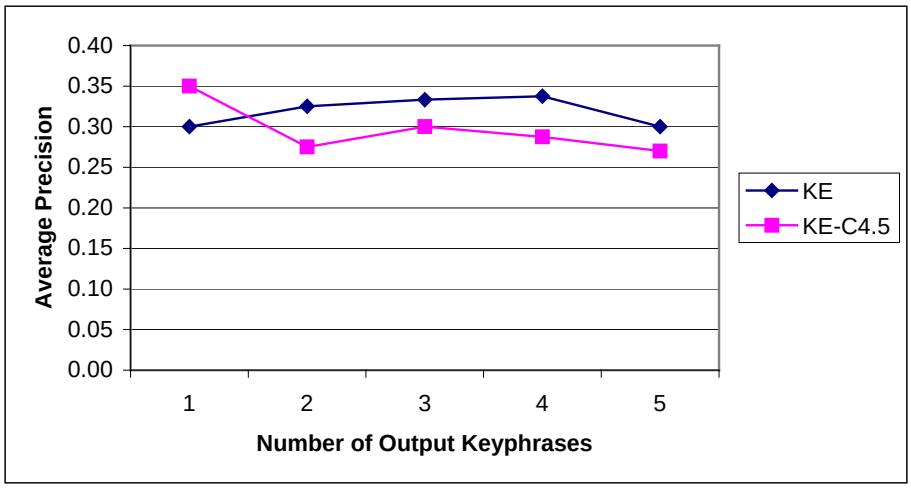

Figure 4.7: Comparison of KE and KE-C4.5

Although KE gives better results when it is tuned by neural networks, neural networks have been criticized for their poor interpretability (i.e. the level of understanding and insight provided by the model). It is difficult to extract classification rules from neural networks. The C4.5 learning algorithm, however, can do that easily [40]. Although the performance of KE-C4.5 is not as good as that of KE, KE-C4.5 can help us to understand how a phrase has been classified as a keyphrase or a non-keyphrase in this experiment. Figure 4.8 and Figure 4.9 show the decision trees constructed from the term set and the term phrase set respectively. Decision nodes are represented by rounded rectangles. All the attributes have been normalized (for details, see Section 3.4), so they lie in the range of 0 to 1.

The process of term classification is simplified by seven rules (see Figure 4.8). Terms, that appear at the beginning of the document, appear in the title, and have been tagged as proper noun, are useful for identifying keyphrases. $TF \times IDF$, however, is trickier; it depends on the values of other attributes, but, in general, large $TF \times IDF$ values are not preferred.

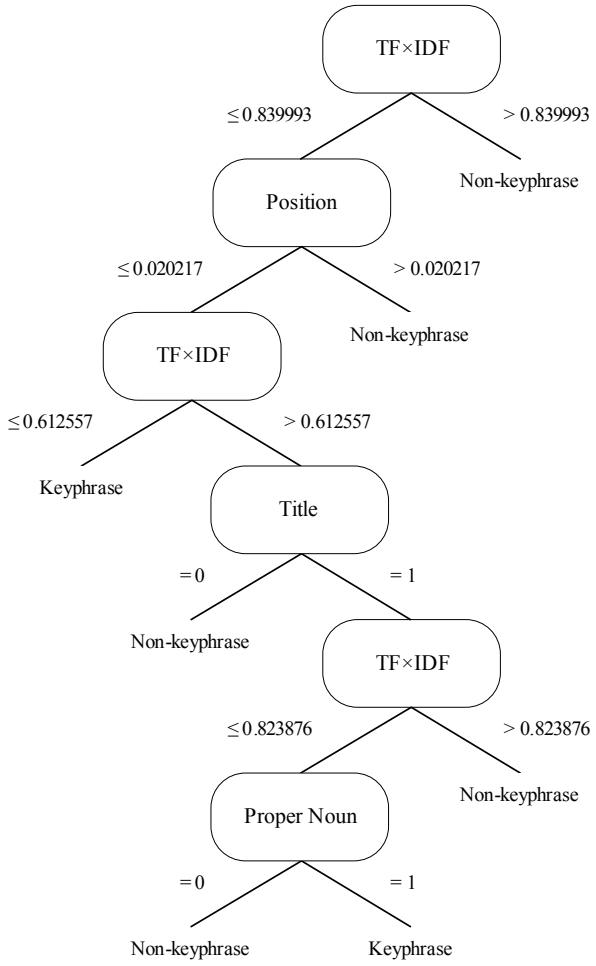

Figure 4.8: Decision tree for classifying terms

The process of term phrase classification is also simplified by seven rules (see Figure 4.9). Term phrases, that appear at the beginning of the document, appear in the title, or not in the title but contain more than one term, are useful for identifying keyphrases. Large *TF×IDF* values are again not preferred.

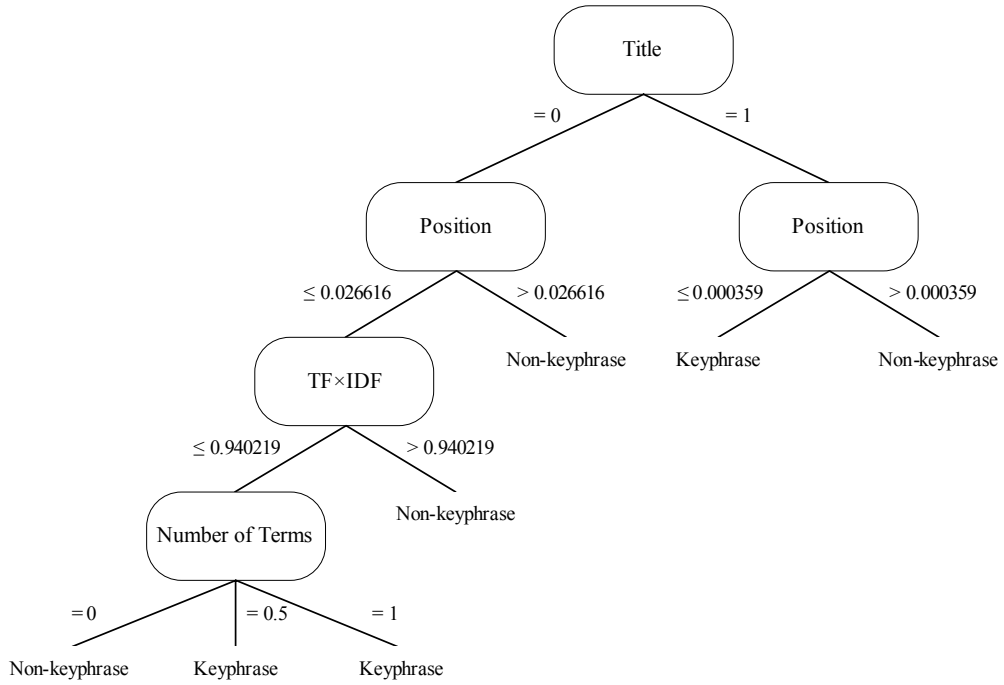

Figure 4.9: Decision tree for classifying term phrases

4.4.7 Different Corpus

To ensure comparability, KE uses the same set of training and testing documents as in GenEx and Kea. Nevertheless, we would like to see how KE performs when it is tested on a different, larger corpus. For convenience, we use *Corpus B* to refer to this corpus, and *Corpus A* to refer to the set of documents used in GenEx and Kea and in our previous experiments.

Corpus B is used to evaluate the generalization performance of KE (tuned by neural networks using the 55 training documents in Corpus A). For details of these documents, see Section 4.2. Corpus A and Corpus B are disjoint. Corpus B contains 231 articles from four journals. These journals cover a variety of subject areas, including life sciences, mathematical sciences, and social sciences. Please see Table 4.20 for the sources of Corpus B. All these articles contain keyphrases supplied by the authors.

To evaluate the correctness of the output keyphrases, we need a set of documents which contain author-assigned keyphrases. Journal articles are, as far as we know, the main source of these kinds of documents. It is not easy to find documents with author-assigned keyphrases in other areas. Even if some documents do contain keyphrases, the quality of these keyphrases might not be as good as that of the keyphrases in journal articles. For example, keyphrases could be found in the *meta* tag of some web pages. However, these phrases are often unreliable and misleading, so most major search engines, including AltaVista, have stopped using them [97]. A recent study also confirms that the importance of these phrases to search engine ranking is little [31]. Therefore, journal articles have been used in this experiment (and the experiment discussed in Section 5.6).

Table 4.20: Sources of Corpus B

Journal Name	Field	Number of Documents
Journal of Molecular Biology	Molecular Biology	46
Information and Software Technology	Information Systems	65
Journal of Economic Behaviour and Organization	Economics and Econometrics	57
International Journal of Educational Development	Education	63
All		231

Table 4.21 shows that, on average, the documents in Corpus B are about 80% longer than the testing documents in Corpus A. For details of the length of the testing documents, see Table 4.2.

Table 4.21: Number of words per document (Corpus B)

Journal Name	Average	Range	Standard Deviation
Journal of Molecular Biology	8539.28	4777.00-15140.00	2316.76
Information and Software Technology	7591.46	2922.00-12676.00	2202.08
Journal of Economic Behaviour and Organization	7488.79	3168.00-13187.00	2293.25
International Journal of Educational Development	8258.63	2567.00-16311.00	2359.49
All	7936.83	2567.00-16311.00	2316.26

Table 4.22 shows that the documents in Corpus B contain much fewer keyphrases than those in Corpus A. On average, there are only 4.58 keyphrases per document in Corpus B compared with 7.46 in Corpus A (for details, see Table 4.3).

Table 4.22: Number of keyphrases per document (Corpus B)

Journal Name	Average	Range	Standard Deviation
Journal of Molecular Biology	4.83	4.00-5.00	0.38
Information and Software Technology	4.35	3.00-7.00	0.94
Journal of Economic Behaviour and Organization	4.39	2.00-8.00	1.11
International Journal of Educational Development	4.79	3.00-9.00	1.12
All	4.58	2.00-9.00	0.98

Similar to Corpus A, most of the keyphrases in Corpus B contain one to two words (see Table 4.23). Because of the nature of molecular biology, three documents in the Journal of Molecular Biology provide non-alphanumeric keyphrases (e.g. ß-actin mutants and ß-sheet), and 15 provide keyphrases containing numbers (e.g. HIV-1 fusion and T7 RNA polymerase).

Table 4.23: Number of words per keyphrase (Corpus B)

Journal Name	Average	Range	Standard Deviation
Journal of Molecular Biology	1.68	1.00-2.40	0.35
Information and Software Technology	2.00	1.00-2.60	0.36
Journal of Economic Behaviour and Organization	1.69	1.00-2.40	0.33
International Journal of Educational Development	1.70	1.00-2.33	0.32
All	1.78	1.00-2.60	0.36

Table 4.24 shows that, on average, 88% of the keyphrases in Corpus B can be found in the document, which is slightly higher than the 82% in Corpus A. This confirms the findings by Turney [105] that 70-90% of keyphrases appear somewhere in the document.

Table 4.24: Percentage of keyphrases found in the document (Corpus B)

Journal Name	Average	Range	Standard Deviation
Journal of Molecular Biology	0.92	0.60-1.00	0.14
Information and Software Technology	0.87	0.25-1.00	0.19
Journal of Economic Behaviour and Organization	0.88	0.25-1.00	0.18
International Journal of Educational Development	0.85	0.40-1.00	0.16
All	0.88	0.25-1.00	0.17

Table 4.25 shows the performance of KE on different journals in Corpus B. KE does not seem to perform well in Corpus B compared with Corpus A. We believe this is because of the higher compression (or document-keyphrase) ratio in Corpus B. On average, the documents in Corpus B are longer (see Table 4.21) but the keyphrase lists are shorter (see Table 4.22) than the testing documents in Corpus A. KE gives similar performance results on these journals, except when the desired number of output keyphrases is set to two. The average precision of these journals lies mostly around 0.20. The experiments confirm the domain independence of KE: KE successfully extracts keyphrases from documents on different subject areas (in Corpus B) while it has been trained on something totally different (i.e. training set in Corpus A). Figure 4.10 shows the comparison of the performance of KE on Corpus B with varying number of output keyphrases.

Table 4.25: Performance of KE on Corpus B

Journal Name	Number of Keyphrases	Average Number of Correct Keyphrases	Standard Deviation
Journal of Molecular Biology	1	0.20	0.40
	2	0.39	0.58
	3	0.61	0.71
	4	0.83	0.77
	5	0.87	0.83
Information and Software Technology	1	0.17	0.38
	2	0.31	0.53
	3	0.58	0.68
	4	0.77	0.79
	5	0.94	0.88
Journal of Economic Behaviour and Organization	1	0.19	0.40
	2	0.49	0.66
	3	0.65	0.74
	4	0.79	0.80
	5	0.93	0.84
International Journal of Educational Development	1	0.19	0.43
	2	0.44	0.56
	3	0.60	0.68
	4	0.84	0.81
	5	0.98	0.91
All	1	0.19	0.40
	2	0.41	0.58
	3	0.61	0.70
	4	0.81	0.79
	5	0.94	0.86

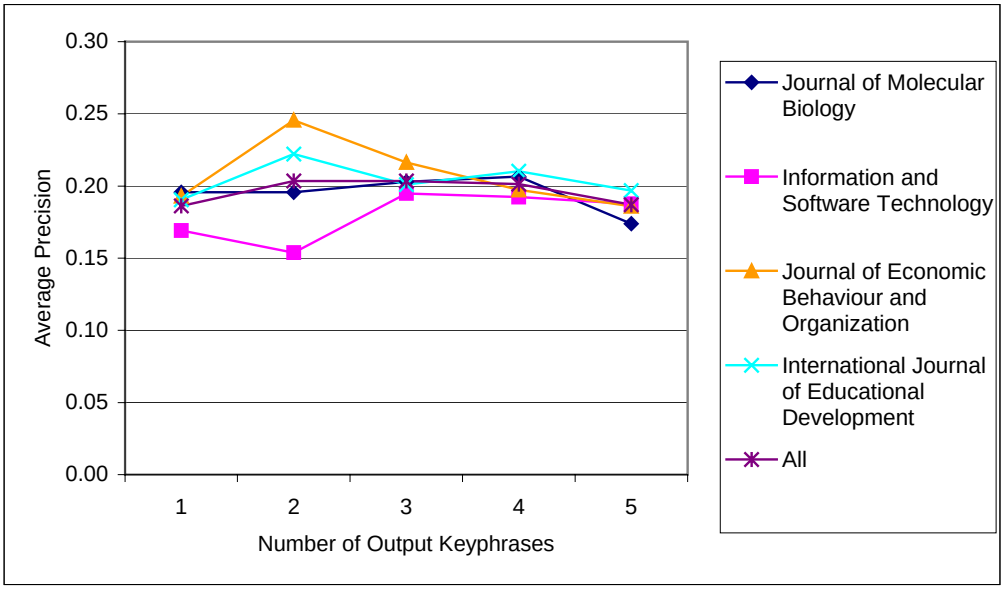

Figure 4.10: Comparison of the performance of KE on different journals

4.5 Discussion of Results

The above performance numbers are misleadingly low. Author-assigned keyphrases are often a small subset of the set of good quality keyphrases for a given document. On average, there are only 7.46 keyphrases per document in Corpus A (both training set and testing set) and 4.58 in Corpus B, and these phrases constitute less than 1% of the document length. In contrast, the desired number of key sentences is usually defined to be 15-20% of the document length or a maximum of four sentences [112]. A more accurate picture can be obtained by asking human assessors to evaluate the machine-extracted keyphrases. GenEx has been tested on 267 web pages: 62% of the keyphrases extracted from these pages are rated by human assessors as 'good', 18% as 'bad', and 20% as 'no opinion'. This suggests that about 80% of the keyphrases extracted by GenEx are acceptable, which should be sufficient for many applications [106].

Some of the machine-extracted keyphrases are rather close to their corresponding author-assigned keyphrase, but because of the stemmer employed, they are regarded as different. For example, the author-assigned keyphrase 'cell assembly' is considered different from the machine-extracted keyphrase 'cell assemblies' because the stemmer maps 'assembly' to 'assemb' and 'assemblies' to 'assembl' (see the first example in Table 4.18). However, this kind of problem is inevitable if an automatic performance measure is used.

We notice that some common words are ranked fairly high in the output list despite the use of stopword lists and IDF. These words come from two main categories. Recall that the score of a term (or term phrase) is dependent on *TF*×*IDF*, *position*, and other attributes. Terms such as 'chapter' tend to occur at the beginning of the document. Early occurrence often boosts the score of these terms and increases the likelihood that they are output, though their IDF might be low. In addition, because of the nature of Corpus A, terms such as 'person', which tend to occur rather frequently in everyday documents, appear only in a few documents of the corpus. This boosts the IDF of these terms and improves their ranking. A possible way of solving this problem is to add these common words to the stopword lists, but this will make KE more domain-dependent, and that is not what we want.

The use of *proper noun* appears to degrade the performance of KE. This is probably because the training and testing documents are all academic papers, which tend to contain many proper nouns, especially in the References section. Indicator phrases may be used to resolve this problem by ignoring all the words in the References section, but this will make KE more domain-dependent. However, we expect that proper nouns might be useful in some domains (e.g. news) where they tend to occur less frequently, but further testing is needed to support this.

Syntactic methods (e.g. the use of italics) [57] seemed helpful in extracting high quality keyphrases, and initially they were considered as an attribute for keyphrase extraction. However, all the documents in Corpus A are in ASCII and Unicode format, so we cannot implement this.

4.6 Summary

This chapter presents the experimental results. We have discussed the corpus used to train and test KE, and the criteria used for evaluating the output keyphrases. The comparison of the individual performance of different attributes, the performance of different combinations of attributes and *TF*×*IDF*, the performance of different keyphrase extraction tools, and the performance of KE on different learning methods and different corpora have also been presented in this chapter. The next chapter will extend the use of KE to another language.

CHAPTER FIVE

Extraction of Keyphrases from Chinese Documents

5.1 Overview

This chapter extends the use of the KE algorithm to Chinese documents. To explore the use of KE in another language, we make some changes to KE and apply it to Chinese documents. The changes are kept minimal because we have to make sure that they will not affect the main features of KE while adapting KE to another language. For details of these changes, see Section 5.4. This variation of KE has been tested on a set of Chinese documents on different subject areas, and the experiments indicate that it works on these documents. This confirms the domain independence of KE and suggests that KE can be applied to another language for keyphrase extraction. KE is, as far as we know, the first keyphrase extraction algorithm that has been validated and evaluated on different domains and different languages.

Section 5.2 provides background information about the adaptation of KE to another language. Section 5.3 defines some important terms used in this chapter. Section 5.4 describes how KE has been extended to extract keyphrases from Chinese documents. Section 5.5 discusses the corpus and criteria used for evaluating the output keyphrases. Section 5.6 evaluates the performance of KE on Chinese documents. The performance results are discussed in Section 5.7. Section 5.8 concludes this chapter.

5.2 Background

The experiments discussed in Section 4.4 indicate that the KE algorithm can be used to extract keyphrases from heterogeneous documents. These documents, however, are all English documents. To explore and adapt KE to another language, we make minimal modifications to KE and extend it to KEC (i.e. a variation of KE for Chinese documents). Because of a different language, we use a different corpus (to train and test KEC) and different criteria (to evaluate the performance of KEC). KEC has been trained and tested on a set of Chinese documents. The experiments show that KEC successfully extracts keyphrases from these documents. This is, as far as we know, the first time a keyphrase extraction algorithm has been validated and evaluated on different domains and different languages.

Five attributes are used in KE: *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. *TF×IDF* is used because keyphrases tend to occur frequently in the document and concentrate in a few documents of the collection. *Position* is used because keyphrases tend to occur at the beginning of the document. *Title* is used because keyphrases often occur in the title of the document. *Proper noun* is used because some keyphrases are proper nouns. *Number of terms* is used because most keyphrases contain more than one word. These attributes characterize keyphrases and are independent of languages (except for IDF), so all of them should be able to be used in another language.

To adapt KE to another language, we need to make three modifications:

- A set of documents (written in that language) is needed to train and test the modified algorithm. Testing documents are used to evaluate the performance of this algorithm on that language. Training documents are used to train this algorithm and to calculate the IDF of a term; if the training documents and the document that contains this term are in different languages, we will not be able to measure the rarity of this term across the training set.

- A part-of-speech tagger (for that language) is needed to get the syntactic category of the words in the document. This is because only adjectives, verbs, nouns and noun phrases are selected as candidate phrases.

- A stemmer (for that language) is needed to stem the words in the document if stemming applies to that language.

5.3 Definitions

Before we discuss the KEC algorithm, we need to define some important terms. The *character* (though some people would refer to it as a 'word') is the smallest unit of measurement for Chinese documents. A *word* (though some people would refer to it as a 'phrase') consists of one or more characters, e.g. 关系 (relations) is a word consisting of two characters. A *phrase* consists of one or more words, e.g. 国际关系 (international relations) is a phrase consisting of two words. There is no easy way to tokenize a phrase. This is because, unlike English, Chinese is not delimited by white spaces. Therefore, it is possible that different people have different opinions on the number of words in a phrase. For example, some people may suggest that 国际关系 (international relations) is a phrase consisting of only one word. To solve this problem, we use a part-of-speech tagger to tokenize a phrase (the resulting syntactic tags can be ignored in this case); if the tagger breaks a phrase into two words, this phrase consists of two words.

5.4 Extension of KE to Chinese Documents

The user provides KEC with a document, the title of the document, and the desired number of output keyphrases as input. KEC uses a part-of-speech tagger (to tag the document) and a neural network (to tune KEC), and provides a list of keyphrases as output. Unlike KE, KEC does not need a stemmer to stem the document. This is because, unlike English, Chinese is not an alphabetical language and therefore stemming is not applicable to the Chinese language.

It would be ideal if KE (which is trained on English documents) could be used without change to extract keyphrases from Chinese documents. However, since the documents are written in a different language, this is not possible. Therefore, the following changes have been made to KE. We highlight the changes that have been made to each of the steps in KE; things that are the same as in KE are skipped. For a detailed description of KE, see Section 3.3. These changes are kept minimal because we need to make sure that it is KE (and not these changes) which makes the extraction work.

Step: 1. Select Words

Changes: Because of a different language, the stopword list has changed. The new list contains only two stopwords, i.e. 是(is), and 有(have).

Remarks: Everything else is the same as in KE, i.e. select all the words which have been tagged as adjective, verb and noun, and are not included in the stopword list.

Step: 2. Score Words

Changes: Stemming does not apply to the Chinese language. Therefore, there is no need to stem the selected words and detect equivalent stems. Also, since the words are not stemmed, we would refer to them as 'words' rather than 'terms'.

Words that contain only one character, e.g. 人(person) and 字(word), are deleted because they often give little semantic value.

Remarks: Everything else is the same as in KE, i.e. delete words that occur only once in the document, use the same set of attributes to calculate the score of each word, and sort these words in order of score followed by *position*.

Step: 3. Select Phrases

Changes: A noun phrase is defined as zero, one or two nouns, adjectives or verbal nouns/ nominal verbs followed by a noun or a verbal noun/ nominal verb. There is no gerund in Chinese, so at first a noun phrase was defined as zero, one or two nouns or adjectives followed by a noun, but then we noticed in our experiment that some of the words in the author-assigned keyphrases were tagged as 'vn' (which is equivalent to verbal noun/ nominal verb in English), so we tried to include this syntactic tag in the definition of noun phrases. Some of the words in Chinese have both nominal and verbal properties (this also happens in English), e.g. 供应 (supply), and are therefore tagged as 'vn'. The experimental results indicate that if we include verbal nouns/ nominal verbs in the definition of noun phrases, the performance of KEC improves by -5%, -3%, 0%, 4% and 7% when the desired number of output keyphrases is set to one, two, three, four and five respectively.

Remarks: Everything else is the same as in KE, i.e. select all the noun phrases in the document.

Step: 4. Score Phrases

Changes: Stemming does not apply to the Chinese language. Therefore, there is no need to stem the selected phrases and detect equivalent stem phrases. Also, since the phrases are not stemmed, we would refer to them as 'phrases' rather than 'term phrases'.

Phrases that contain only one character are deleted because they often give little semantic value.

Remarks: Everything else is the same as in KE, i.e. delete phrases that occur only once in the document, use the same set of attributes to calculate the score of each phrase, and sort these phrases in order of score followed by *position* and *number of terms*.

Step: 5. Expand Words

Changes: –

Remarks: Everything is the same as in KE, i.e. for each word, find all the phrases that contain the word, and link it with the highest scoring phrase.

Step: 6. Drop Duplicates

Changes: –

Remarks: Everything is the same as in KE, i.e. for each phrase, link it with the highest scoring word.

Step: 7. Display Output

Changes: Since stemming is not performed in KEC, there is no need to find the 'corresponding' phrases in the input document.

Remarks: Everything else is the same as in KE, i.e. delete subphrases if they do not perform better than their superphrases.

5.5 Corpus and Evaluation Criteria

Because of the vast availability of English documents (both online and offline), Olsson *et al.* [75] use English documents to train their topic classification method for Czech documents. We could also do this if IDF was not used in the KEC algorithm. However, because of the language-dependent nature of IDF (which calculates the number of documents a word/ term occurs in) and the fact that KEC is intended for Chinese documents and KE is trained on English documents, a set of Chinese documents have been used for the training of KEC.

We use the documents in [77] to build the corpus for training and testing the KEC algorithm (for convenience, we use *Corpus C* to refer to this corpus). This website is chosen because it provides a set of journal articles covering a wide range of topics including economics, politics, law, education, literature, technology, etc. The wide diversity of these topics also means that we can confirm the domain independence of KE in this experiment. For details of the reasons why journal articles are used, see Section 4.4.7. All the articles in [77] are written in simplified Chinese characters and only those with author-assigned keyphrases are selected in Corpus C.

Three part-of-speech taggers [91, 100, 122] were considered and we decided to use [122] to tag the journal articles. [91] is intended for documents written in simplified Chinese characters, but it is not as popular as [100, 122]. [100], though more popular, is designed for traditional Chinese characters, but all the documents in Corpus C are written in simplified Chinese characters. [122] is a popular tagger intended for simplified Chinese characters, and is therefore chosen.

As mentioned before, Corpus C is used to evaluate the performance of KE on Chinese documents and to confirm the generalization performance of KE. Corpus C is disjoint from Corpus A and Corpus B. It contains 265 documents; 50 of these documents have been randomly selected and used to train KEC, and the remaining documents have been used to test KEC.

Since stemming is not applicable to the Chinese language, the criteria used for evaluating the output keyphrases in this experiment have changed: a machine-extracted keyphrase is said to be correct if it matches an author-assigned keyphrase.

Table 5.1 shows that, on average, the documents in the training set are longer than those in the testing set.

Table 5.1: Number of characters per document (Corpus C)

Corpus	Average	Range	Standard Deviation
Training	14957.14	5272.00-37124.00	7432.15
Testing	13876.68	5486.00-36321.00	6091.25

Table 5.2 shows that the documents in the training set contain slightly fewer keyphrases than those in the testing set. On average, there are only 3.78 keyphrases per document in Corpus C compared with 7.46 in Corpus A and 4.58 in Corpus B.

Table 5.2: Number of keyphrases per document (Corpus C)

Corpus	Average	Range	Standard Deviation
Training	3.56	2.00-5.00	0.61
Testing	3.83	2.00-8.00	0.91

Table 5.3 shows that most of the keyphrases in Corpus C contain three to four characters.

Table 5.3: Number of characters per keyphrase (Corpus C)

Corpus	Average	Range	Standard Deviation
Training	3.73	2.00-8.00	1.14
Testing	3.87	2.00-6.00	0.84

Table 5.4 shows that over 94% of the keyphrases in Corpus C can be found in the document, which is much higher than the 82% in Corpus A and the 88% in Corpus B.

Table 5.4: Percentage of keyphrases found in the document (Corpus C)

Corpus	Average	Range	Standard Deviation
Training	0.94	0.50-1.00	0.13
Testing	0.96	0.50-1.00	0.10

5.6 Experimental Results

This section describes how we train the KEC algorithm and evaluate the performance of KEC on Chinese documents.

5.6.1 Training

The KEC algorithm has to be trained before it can be applied to new documents for keyphrase extraction. During training, a total of 120 phrases (and words), including keyphrase and non-keyphrase examples (and keyword and non-keyword examples), are selected from each training document.

Like KE, the set of words (i.e. output of Step 2 in KEC) and the set of phrases (i.e. output of Step 4 in KEC) were tuned separately by a back-propagation neural network. The resulting sets were then combined to perform Step 5, 6 and 7 of the KEC algorithm.

Keyphrase and non-keyphrase examples are selected in the same way as in KE, but keyword and non-keyword examples are selected differently. To get keyword examples from the document, the author-assigned keyphrases are broken into words using a part-of-speech tagger (rather than white spaces as in KE, see Section 4.4.1). This is because an author-assigned keyphrase may contain several words, which means an author-assigned keyphrase could provide more than one keyword example. The resulting words are used as keyword examples, and words (other than those in the author-assigned keyphrases) are randomly selected from the document as non-keyword examples.

5.6.2 Different Language

Table 5.5 shows the performance of KEC on Corpus C. KEC does not seem to perform well (the average precision of KEC lies between 0.12 and 0.16) compared with KE on Corpus A and Corpus B (see Section 4.4.5 and Section 4.4.7). There is also a tendency for the average precision of KEC to fall. However, considering there are only 3.78 keyphrases per document in Corpus C (see Table 5.2), the relatively poor performance is actually quite understandable. The experimental results suggest that KE can be used to extract keyphrases from Chinese documents. The results also confirm, in addition to the experiment discussed in Section 4.4.7, the domain independence of KE. Figure 5.1 shows the performance of KEC on Chinese documents with varying number of output keyphrases.

Table 5.5: Performance of KEC on Corpus C

Algorithm	Number of Keyphrases	Average Number of Correct Keyphrases	Standard Deviation
KEC	1	0.16	0.37
	2	0.27	0.51
	3	0.38	0.66
	4	0.50	0.73
	5	0.58	0.78

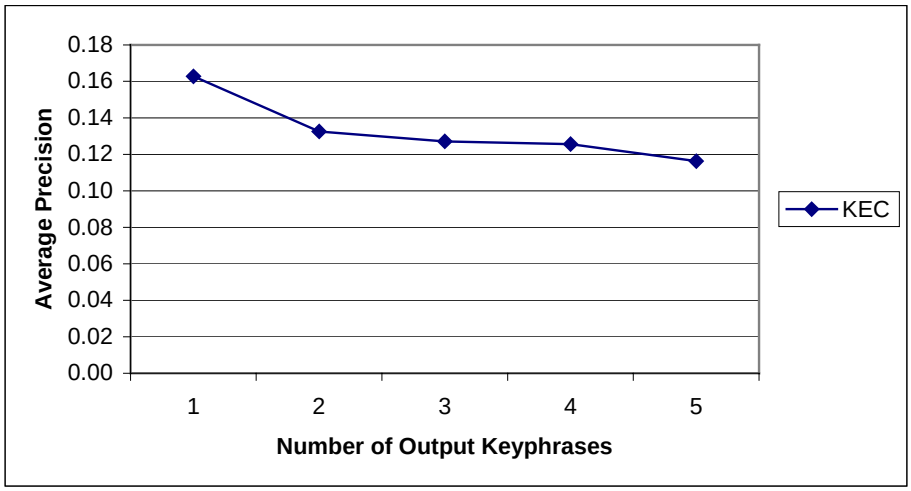

Figure 5.1: Performance of KEC on Chinese documents

5.7 Discussion of Results

We notice that the syntactic pattern of the author-assigned keyphrases in Chinese documents is much more complicated than that in English documents. For example, 食物安全 (food safety) is an author-assigned keyphrase, in which 食物 (food) is tagged as noun and 安全 (safety) as nominal adjective. However, we are not sure whether this is because of the part-of-speech tagger employed or the nature of the Chinese language. Only 74% of the author-assigned keyphrases in the testing documents (and 78% in the training documents) follow the definition of noun phrases in KEC. This means that KEC will not be able to provide 26% of the author-assigned keyphrases in the testing documents as output if this definition is used to select noun phrases from these documents. To improve the performance of KEC, we could use an ad hoc definition of noun phrases by analyzing the syntactic tags of all the author-assigned keyphrases and defining noun phrases using these tags, or we could (like the LAKE algorithm, see Section 2.2.3) define a considerable number of manually predefined linguistics-based patterns. However, these will make KE more language-dependent and reduce the adaptability of KE to another language. Since KE contains several language-dependent components (e.g. a stemmer, a part-of-speech tagger, and a set of training and testing documents, see Section 5.2), it is already language-dependent. However, we want KE to be language-independent as much as possible, and thus limit these components to a minimum. We only use these components as user-defined parameters to the main language-independent part of KE. These components are necessary for KE to work in another language. In addition, they help to increase accuracy.

A stemmer reduces variants of a word to a single form, and sometimes this helps to improve the performance of KE on English documents. Although a different phrase (from the corresponding author-assigned keyphrase) might be output, as long as the stem of this phrase matches the stem of an author-assigned keyphrase, it is considered correct (see Section 4.3.1 for the evaluation criteria used in English documents). However, this is not the case for Chinese documents. Since stemming does not apply to the Chinese language, a machine-extracted keyphrase needs to be exactly the same as an author-assigned keyphrase; if not, it is considered incorrect (see Section 5.5 for the evaluation criteria used in Chinese documents). For example, 工业化 (industrialization) is provided as an author-assigned keyphrase. If it is in an English document, phrases such as 'industry' will be considered correct (in addition to 'industrialization'). However, if it is in a Chinese document, the output keyphrase has to be exactly '工业化'; anything else will be considered incorrect. This shows the additional difficulty of extracting keyphrases from Chinese documents.

Sometimes, KEC fails to combine several words (or phrases) together as an output keyphrase. For example, the author provides 教育体制改革 (education system reform) as a keyphrase, but instead of outputting this phrase, KEC breaks it into three words, i.e. 教育 (education), 体制 (system) and 改革 (reform) and outputs these words as keyphrases. However, the decision of whether to divide a phrase is entirely made by the computer, so there is not much we can do. Over-grouping of words (or phrases) could as well create problems as some authors might prefer to provide more but shorter keyphrases, e.g. 教育体制 (education system) and 改革 (reform).

Some of the machine-extracted keyphrases are more general (or more specific) than the author-assigned keyphrases provided. For example, KEC outputs 教育 (education) as a keyphrase, but the author-assigned keyphrase list does not contain this phrase (probably because the author thinks this phrase is too general), though the document is concerned with education policy (so it would be sensible to provide education as a keyphrase). This also happens in English documents, see the third example in Table 4.18. However, this kind of problem is inevitable if author-assigned keyphrases are used as a standard of comparison for evaluating the output keyphrases.

5.8 Summary

This chapter extends the use of the KE algorithm to Chinese documents. We have explained why KE can be adapted to another language and how this is done, and extended KE (to KEC) to extract keyphrases from Chinese documents. In addition, the corpus and criteria used for evaluating the performance of KEC have been discussed in this chapter. We have also presented the performance results of KEC on Chinese documents. The next chapter will discuss the conclusions and future work.

CHAPTER SIX
Conclusions

6.1 Overview

This chapter summarizes this thesis and the contributions of our research work, and suggests two possible ways to improve the performance of KE.

Section 6.2 summarizes this thesis. Section 6.3 discusses the contributions of our work. Section 6.4 discusses two areas which warrant further investigation.

6.2 Summary

This thesis proposes a new domain-independent keyphrase extraction algorithm called KE. KE is not tied to a specific domain; it is designed to summarize a given document, which can be on any topic, in a few keyphrases automatically extracted from the body of that document.

KE approaches the problem of keyphrase extraction as a classification task: a document is seen as a set of phrases, and KE should correctly classify a phrase as a keyphrase or a non-keyphrase. Five attributes have been found useful to do this in our experiments: *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. The weights associated with these attributes are tuned by a neural network using examples of keyphrases and non-keyphrases during training. After training, they are frozen and used by KE to classify phrases in new documents.

The KE algorithm consists of seven steps which could be grouped into three activities:

1. KE uses a part-of-speech tagger to select words and phrases from the input document, and a vector of terms and a vector of term phrases to represent this document. Each term is characterized by four attributes: *TF×IDF*, *position*, *title*, and *proper noun*. A score is assigned to each term based on these attributes. The term phrase vector is similar to the term vector, except that each term phrase is characterized by a different set of attributes: *TF×IDF*, *position*, *title* and *number of terms*.

2. A one-to-one relationship is established between the terms and the term phrases. For each term, KE finds all the term phrases that contain the term, and link it with the highest scoring term phrase. More than one term may link to the same term phrase. If that is the case, the term phrase will be linked to the highest scoring term. The result is a list of term phrases ordered by the scores of the corresponding terms.

3. The list of term phrases is then used to generate the output keyphrases. For each term phrase in the list, KE finds the most frequent corresponding phrase in the document. The result is a list of phrases. If a phrase is a subphrase of another phrase, it will only be accepted as a keyphrase if it is ranked higher; otherwise, it will be deleted from the output list.

KE is based on GenEx and Kea, but differs from them in several ways.

- Purely statistical methods are used in GenEx and Kea. KE, however, uses a combination of statistical and text processing techniques for keyphrase extraction. Part-of-speech tagging has been used to improve the quality of candidate phrases. Better quality data (i.e. better quality candidate phrases) often lead to better performance results. Only adjectives, verbs, nouns and noun phrases are selected as candidate phrases.

- KE uses a different set of attributes to discriminate between keyphrases and non-keyphrases: *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. Kea uses only two attributes: TF×IDF and distance. GenEx, on the other hand, uses many more attributes, but it does not use *TF×IDF* and *title*. Location-based methods are used in all these algorithms. *TF×IDF* is used because keyphrases tend to occur frequently in the document and concentrate in a few documents of the collection. *Title* is used because keyphrases often occur in the title of the document. *Proper noun* is used because some keyphrases are proper nouns. *Number of terms* is used because most keyphrases contain more than one word.

- KE uses a different machine learning algorithm; it is tuned by a neural network. Neural networks are used because they provide a simple black box for pattern recognition and have proved useful in some information retrieval (IR) projects. GenEx is tuned by a genetic algorithm, while Kea is based on the naïve Bayes learning technique.

- KE is a different model; it consists of seven steps, and takes both words and phrases as candidate phrases. Words are used for ranking purposes. It is generally preferable to represent documents and measure the importance of each representation element using words (or terms, to be precise). Phrases, on the other hand, are used for output purposes. This is because documents are summarized by a set of phrases, not words. Kea is a simple model; it only selects phrases as candidate phrases, so it does not involve any linking between words and phrases. GenEx is more complicated; it consists of ten steps, considers both words and phrases, and involves many post-processing tasks.

The experimental results suggest that these differences make KE a better algorithm than either GexEx or Kea.

KE has been tested on two different corpora. The first corpus is the same as the one used in GenEx and Kea, and it has been used to train and test KE in all our experiments (except the one discussed in Section 4.4.7). The criteria used for evaluating the output keyphrases are also the same as in GenEx and Kea, so direct comparison is possible. The second corpus is different and larger than the first one, and it has been used to test the generalization performance of KE. The evaluation criteria used for this corpus are the same as for the first corpus.

We have evaluated the individual performance of different attributes and the performance of different combinations of attributes. The experiments suggest that *position* gives the best individual performance and that the best combination of attributes involves *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. In addition, we have compared different combinations of *TF×IDF*, and found that the standard TF and Kea's IDF gives the best performance. The experiments also indicate that KE performs better than other keyphrase extraction tools, including GenEx and Kea, and that it significantly outperforms Microsoft Word 2000's AutoSummarize feature. We have tried using the C4.5 decision tree learning method to tune KE, but the experiments show that neural networks are better for keyphrase extraction than this method. The domain independence of KE has also been confirmed in our experiments using the second corpus.

To extend the use of the KE algorithm to Chinese documents, we have made minimal changes to KE and extended it to KEC. Because of a different language, we use a different corpus (to train and test KEC) and different criteria (to evaluate the performance of KEC). KEC has been trained and tested on a set of Chinese documents. The experiments show that KEC successfully extracts keyphrases from these documents. This is, as far as we know, the first time a keyphrase extraction algorithm has been validated and evaluated on different domains and different languages.

6.3 Contributions

A number of journal and conference papers have been published based on the material discussed in this thesis [66-69]. The contributions of our work are as follows:

- Proposed a new domain-independent keyphrase extraction algorithm called KE, which is based on GenEx and Kea and uses a combination of statistical and text processing techniques, a different set of attributes, and a different machine learning method to extract keyphrases from documents.

- Evaluated a number of different attributes in our experiments, and found that five attributes are useful for keyphrase extraction: *TF×IDF*, *position*, *title*, *proper noun* and *number of terms*. The usefulness of these attributes has also been confirmed in our experiments using the decision trees generated by the C4.5 learning method.

- Found in our experiments that better results can be achieved if two terms having the same score are ranked in ascending order of *position*, and if two term phrases having the same score are ranked in ascending order of *position* followed by descending order of *number of terms*.

- Evaluated the individual performance of different attributes in our experiments, and found that location-based methods give the best performance result.

- Evaluated the performance of different combinations of *TF×IDF* in our experiments, and found that KE gives the best performance when the standard TF and Kea's IDF is used.

- Found in our experiments that KE performs better than other keyphrase extraction tools and that it significantly outperforms Microsoft Word 2000's AutoSummarize feature (on the problem of keyphrase extraction).

- Found in our experiments that neural networks are better for keyphrase extraction than the C4.5 decision tree learning method.

- Evaluated the generalization performance of KE on a different, larger corpus (than the one used in GenEx and Kea) in our experiments. This corpus contains documents on different subject areas. The domain independence of KE has been validated by testing KE on this corpus. This suggests that KE can be applied to other subject areas for keyphrase extraction.

- Extended the use of KE to Chinese documents. Minimal changes have been made to KE to adapt it to another language. This variation of KE has been tested on a set of Chinese documents on different subject areas, and the experiments indicate that it works on these documents. This confirms the domain independence of KE and suggests that KE can be applied to other languages for keyphrase extraction.

- Showed in our experiments that the attributes and techniques used in KE are useful for keyphrase extraction and can therefore serve as a useful starting point for new keyphrase extraction algorithms. KE is, as far as we know, the first keyphrase extraction algorithm that has been validated and evaluated on different domains and different languages.

- Showed in our experiments that the choice of a learning method and the compression (or document-keyphrase) ratio affect the performance of a keyphrase extraction algorithm.

6.4 Future work

This section discusses two ways to improve the quality of the output keyphrases.

6.4.1 Relevance Feedback

Relevance feedback is a useful technique for improving the effectiveness of an IR system. The idea underlying relevance feedback is simple: it uses responses from the user to mark retrieved documents as relevant or nonrelevant. When the user provides the IR system with a query, the system retrieves documents from the collection based on the query. Relevance information can be obtained by presenting those documents to the user for judgement as relevant or nonrelevant. This information could then be used to reformulate the query. This process of obtaining relevance information and using it in a further search is known as relevance feedback [20].

Relevance feedback helps to improve retrieval performance by choosing important terms from documents which have been identified by the user as relevant and enhancing the importance of these terms in a new query formulation. The effect of such a query alternation process is to 'move' the new query in the direction of the relevant documents and away from the nonrelevant documents, in the hope of retrieving more relevant documents and fewer nonrelevant documents in a later search [87]. Relevance feedback has proved effective in a number of IR projects [42, 87, 89]. One of these projects suggests that adding as few as 20 well-selected terms to the query could result in performance improvements of over 100% [42].

We believe the idea of relevance feedback can be used to improve the performance of a keyphrase extraction algorithm. We could ask the user to assess the quality of the output keyphrases in a trial run. The information provided could then be used to tune the algorithm by adjusting the weight of different attributes used in this algorithm.

6.4.2 Hyperlinks

The term 'document' has been used to refer to plain text (i.e. no hyperlinks [73] or multimedia elements) in this thesis. It will be interesting to see if the use of web documents can improve the performance of KE. Most web documents contain hyperlinks, which add useful information to the document. The study of these kinds of documents is commonly known as *web mining*.

Web mining involves the use of data mining techniques to automatically discover and extract information from web documents and services [55]. It often refers to three different activities: *web structure mining*, *web usage mining*, and *web content mining* [55, 61]. All these activities qualify as data mining and involve the web, but the actual data being mined and the motivation are different.

- Web structure mining attempts to extract information from the topology of the web, i.e. the links among pages [61]. In its pure form, structure mining does not require information about the content of the pages. Kleinberg's HITS algorithm [14, 15, 35, 53] is an example of web structure mining. It uses the web topology to help to find authoritative pages. For example, if we want to retrieve information about 'Harvard' from the web, Harvard University's website should stand out from other sites because it contains the most relevant information and is therefore considered the most authoritative site on this topic. Although Google has not revealed how its search algorithm works, it is believed that the algorithm involves some sort of link analysis [10].

- Web usage mining attempts to extract information about how the people who traverse those links with their browsers make use of them [61]. Usage mining could be the most interesting area from a business point of view because this is where customer behaviour is revealed. The actual behaviour might sometimes be different from what the developers expect. Web usage mining helps to identify this and improve the site's usability. Bell's SearchLight project [92] is an example of web usage mining. It is based on the finding that 40% of search engine queries are repeated. It helps to reduce search time by providing users with information about other people's search results.

- Web content mining attempts to extract useful information from the text, images, and other forms of content that make up the pages [61, 62]. In its pure form, content mining does not require information about the links between pages. Multimedia data mining [120] is still in its infancy, though much of the content of the web involves sound and pictures. Currently, most content mining is actually text mining [3, 61]. Search engines, and IBM's Intelligent Miner for Text [104] are examples of web content mining. Keyphrase extraction may also be seen as a content mining activity if the document from which keyphrases are to be extracted is a web document [55].

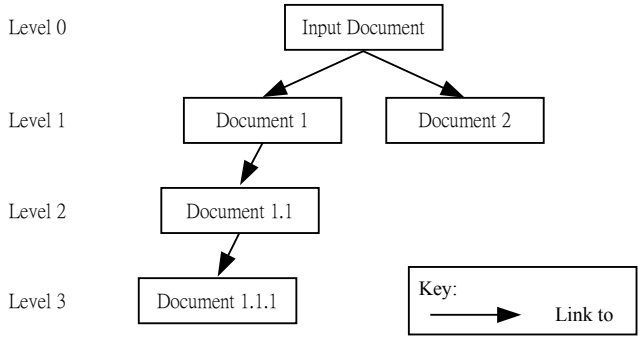

Figure 6.1: Use of hyperlinks for keyphrase extraction

It will be interesting to see if the use of hyperlink information (which has proved effective in HITS and Google) can boost the quality of the output keyphrases. Recall that the performance of a keyphrase extraction algorithm improves significantly when it is trained on documents that are from the same domain as the document from which keyphrases are extracted (for details, see Section 2.2.2). If a hyperlink document is provided, we could follow all the links in that document say to the second level (see Figure 6.1). We will then get a set of documents which is likely to be in the same domain as the input document. The documents linked to the input document should be related to the input document in some way; if not, they would not be linked. The document set could be used to help to identify high quality keyphrases. This way, we would not have to sacrifice domain independence for performance. Also, the algorithm would not be tied to a specific domain because the document set is generated only when a document is provided.

APPENDIX

Formal Specification of KE

7.1 Overview

This appendix describes the KE algorithm using the Z notation. Z is a formal specification language based on mathematics; because of its mathematical nature, requirements written in Z are precise and unambiguous. This is why we use Z to describe KE. This appendix is mostly concerned with the formal specification of KE. To avoid ambiguity, all the steps and attributes involved in KE are specified in Z. To improve readability, these steps are explained with examples and English descriptions.

Section 7.2 discusses the formal specification of KE. Section 7.3 concludes this appendix.

7.2 Formal Description of KE

This section specifies KE in Z and explains the specification with informal English descriptions. Please refer to [47, 96, 116, 117] for further information about the Z notation.

7.2.1 Types

A document is just a string. Although it could be divided up into a sequence of words or even characters, these levels of abstraction are too low. For the purposes of this specification, we suppose that *String* is a basic type:

```
[String]
```

There are 36 part-of-speech tags in the Penn Treebank tagset (for details, see Section 2.7). Nevertheless, we are only interested in adjectives (JJ, JJR, JJS), verbs (VB, VBD, VBG, VBN, VBP, VBZ) and nouns (NN, NNS, NNP, NNPS). Tags other than these are classified as others.

We define *PartOfSpeech* as a free type with 14 constants:

```
PartOfSpeech ::= jj | jjr | jjs | vb | vbd | vbg | vbn |
     vbp | vbz | nn | nns | nnp | nnps | others
```

The set of all valid tagged words is represented as a schema type:

```
┌──── Tag ──────────────────────────────────────
│ s : String
│ p : PartOfSpeech
└──────────────────────────────────────────────
```

Terms are characterized by four attributes: *TF×IDF*, *position*, *title*, and *proper noun*. The set of all valid terms is represented as a schema type:

```
┌──── Term ─────────────────────────────────────
│ str : String
│ tf : ℕ
│ idf : ℕ
│ position : ℕ
│ title : ℕ
│ proper_noun : ℕ
│ score : ℕ
└──────────────────────────────────────────────
```

Term phrases are also characterized by four attributes: *TF×IDF*, *position*, *title* and *number of terms*. The set of all valid term phrases is represented as a schema type:

```
┌──── TermPhrase ──────────────────────────────
│str : String
│tf : ℕ
│idf : ℕ
│position : ℕ
│title : ℕ
│no_of_terms : ℕ
│score : ℕ
└──────────────────────────────────────────────
```

7.2.2 Predefined Functions

When a user provides KE with a document, the document is tagged and stemmed. We use the iterated Lovins stemmer to stem the input document (for efficiency purposes) and the output keyphrases (for evaluation purposes), and Eric Brill's part-of-speech tagger to tag the input document. Since the focus of this appendix is on the specification of KE, the details of how the stemmer and the tagger work can be temporarily ignored. For the purposes of this specification, we suppose that there are two predefined functions performing these tasks. Please refer to Section 2.6 for further information about the stemmer, and to Section 2.7 for the tagger.

```
│stem_ : String ⟶ String
│tag_ : String ⟶ seq Tag
├──────────────────────
│... definitions omitted ...
```

The details of how the following functions work are also not important to us, so we suppose that they are predefined functions (three of these are about string manipulation):

- The function *concat* is used to concatenate a string to the end of another string.

- The function *indexof* is used to get the index of the first occurrence of a string within another string. If it does not occur within that string, zero will be returned.

- The function *scoreterm* is used to calculate the score of a term, taking *TF×IDF*, *position*, *title*, and *proper noun* as input.

- The function *scoretermphrase* is used to calculate the score of a term phrase, taking *TF×IDF*, *position*, *title* and *number of terms* as input.

- The function *tokenize* is used to break a string into tokens (i.e. a sequence of strings).

```
_concat_ : (String × String) ⟶ String
_indexof_ : (String × String) ⟶ ℕ
scoreterm_ _ _ _: (ℕ × ℕ × ℕ × ℕ) ⟶ ℕ
scoretermphrase_ _ _ _: (ℕ × ℕ × ℕ × ℕ) ⟶ ℕ
tokenize_ : String ⟶ seq String
──────────────────────────
... definitions omitted ...
```

7.2.3 Global Constants and Variables

A stopword list contains words with high frequency and little semantic value (for details, see Section 1.3.1). The stopword list used in KE contains 17 common verbs, which are basically the various forms of 'be', 'do', and 'have':

```
Stopwords == { be, were, was, being, am, been, are, is,
   do, did, doing, done, does, have, had, having, has }
```

A corpus is just a collection of documents. We need a corpus to train KE and to calculate the IDF of a term. For efficiency purposes, all the documents in the corpus are stemmed.

```
Corpus == ℙ String
```

The system stems the input document and the title of this document, and stores them in the global variable *document* and *title* respectively. These variables will be used in Section 7.2.6 and Section 7.2.8.

```
document : String
title : String
```

```
Extract
raw_document? : String
raw_title? : String
...
document = stem(raw_document?)
title = stem(raw_title?)
...
```

The system picks all the words tagged as proper noun and stores them in the global variable *proper_nouns*, which will be used in Section 7.2.6.

```
proper_nouns : ℙ String
```

```
getpropernouns_ : seq Tag ⟶ ℙ String
∀ t : seq Tag •
    getpropernouns t =
        { i : 1..#t | (t i).p ∈ {nnp, nnps} •
        stem((t i).s) }
```

```
Extract
raw_document? : String
...
proper_nouns = getpropernouns(tag(raw_document?))
...
```

The following global variables will also be used in the specification (their meaning will become clear as the algorithm is described):

1. The system selects some words (or phrases) from the input document and stores them in the variable *words* (or *phrases*).

2. The system stems the selected words (or phrases) and stores them in the variable *stemmed_words* (or *stemmed_phrases*).

3. The system calculates the score of each term (or term phrase) and stores it in the variable *terms* (or *term_phrases*).

4. The system establishes a one-to-one relationship between the terms and the term phrases, and stores it in the variable *key_term_phrases*.

```
words, phrases : seq String
stemmed_words, stemmed_phrases : String ⟶ String
terms : seq Term
term_phrases : seq TermPhrase
key_term_phrases : iseq String
```

7.2.4 Attributes

Five attributes have been found useful for keyphrase extraction in our experiments (for details, see Section 4.4) and are used in KE: *TF×IDF* (using the standard TF and Kea's IDF), *position*, *title*, *proper noun*, and *number of terms*. The *TF×IDF*, *position*, *title*, *proper noun* and *number of terms* of a term *T* (or a phrase *P*) in a document *D* may be informally defined as:

```
TF = #(tokenize(D) ▷ {T})
IDF = -log #{ c : Corpus | (c indexof T)≠0 ∧ c≠D • c }
Position = (D indexof T) div #tokenize(D)
(title indexof T) ≠ 0 ⇒ Title = 1
(title indexof T) = 0 ⇒ Title = 0
T ∈ proper_nouns ⇒ Proper_Noun = 1
T ∉ proper_nouns ⇒ Proper_Noun = 0
No_Of_Terms = #tokenize(P)
```

These attributes will be revisited and formally defined in Section 7.2.6 and Section 7.2.8.

7.2.5 Selecting Words

Step 1 involves the selection of all the words which have been tagged as adjective, verb and noun, and are not included in the stopword list.

```
selectwords_ : seq Tag ⟶ seq String
─────────────────────────────────
∀ t : seq Tag •
    selectwords t =
         squash { i : 1..#t |
         (t i).p ≠ others ∧
         (t i).s ∉ Stopwords •
         i ⟼ (t i).s }
```

The system tags the input document before passing it to the function *selectwords* and stores the return value of this function in the global variable *words*.

```
┌──── Extract ──────────────────────────────
raw_document? : String
...
──────────────────────────
words = selectwords(tag(raw_document?))
...
└─────────────────────────────────────────
```

7.2.6 Scoring Terms

Step 2 involves six closely related tasks:

1. Stem the selected words
2. Detect equivalent stems
3. Delete terms that occur only once in the document
4. Calculate the *TF×IDF* (using the standard TF and Kea's IDF), *position*, *title* and *proper noun* of each term
5. Assign a score to each term based on these attributes
6. Sort the terms in descending order of score (if two terms have the same score, they are ranked in ascending order of *position*)

The function *stems* performs Task 1, *detectstems* performs Task 2 and 3, *scoreterms* performs Task 4 and 5, and *sortterms* performs Task 6. These functions will be discussed in this section.

The function *stems* takes a sequence of selected words as input and returns a set of string pairs (i.e. a word and its stem). For example, it will return `{ mathematics ⟼ mathemat, mathematician ⟼ mathemat }` if the input document contains 'mathematics' and 'mathematician'.

```
stems_ : seq String ⟶ (String ⟶ String)
─────────────────────────────────────
∀ s : seq String •
    stems s = { i : ran s • i ⟼ stem(i) }
```

The system uses the function *stems* to stem the selected words and stores the return value of this function in the global variable *stemmed_words*.

```
──── Extract ──────────────────────────────
...
─────────────────────────
stemmed_words = stems(words)
...
──────────────────────────────────────────
```

Multiple occurrences of a given stem (i.e. words with the same stem) are combined into a single term. For example, 'mathematics' and 'mathematician' are combined into 'mathemat' if they occur in the input document (see Figure 7.1).

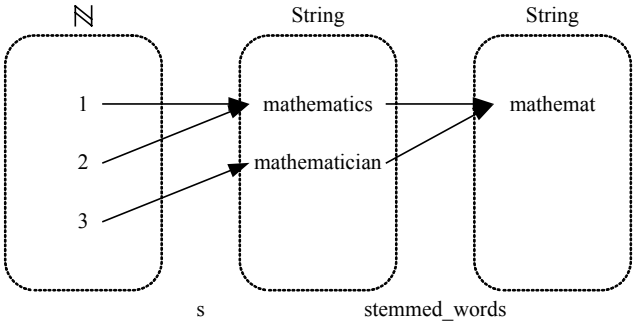

Figure 7.1: Detecting equivalent stems

```
detectstems_ : seq String ⟶ seq String
─────────────────────────────────────
∀ s : seq String •
    detectstems s =
         squash { r : ran(s ⨾ stemmed_words) |
         #((s ⨾ stemmed_words) ⊳ { r }) > 1 •
         min(dom((s ⨾ stemmed_words) ⊳ { r })) ⟼ r }
```

For each term, the value of *TF×IDF*, *position*, *title*, and *proper noun* is calculated. Informal definitions and descriptions of these attributes can be found in Section 1.3.

```
scoreterms_ : seq String ⟶ seq Term
─────────────────────────────────────
∀ s : seq String •
    scoreterms s ⇔ ∀ t : seq Term •
    #s = #t
    ∀ i : 1..#t •
    (t i).str = s i
    (t i).tf =
         #((words ⨾ stemmed_words) ⊳ { (t i).str })
    (t i).idf =
         -log #{ d : Corpus |
         (d indexof (t i).str) ≠ 0 ∧ d ≠ document • d }
    (t i).position = (document indexof (t i).str) div
          #tokenize(document)
    (title indexof (t i).str) ≠ 0 ⇒ (t i).title = 1
    (title indexof (t i).str) = 0 ⇒ (t i).title = 0
    (t i).str ∈ proper_nouns ⇒ (t i).proper_noun = 1
    (t i).str ∉ proper_nouns ⇒ (t i).proper_noun = 0
         ...
```

Each term is scored based on these attributes.

```
scoreterms_ : seq String ⟶ seq Term
─────────────────────────────────────
∀ s : seq String •
    scoreterms s ⇔ ∀ t : seq Term •
         ...
    ∀ i : 1..#t •
         ...
    (t i).score =
         scoreterm((t i).tf*(t i).idf, (t i).position,
         (t i).title, (t i).proper_noun)
```

The function *sortterms* sorts the objects in the sequence by score followed by *position*. If object *a* has a higher score than object *b*, *a* is nearer the top of the sequence; if *a* has the same score as *b*, the object with a smaller *position* value is nearer the top.

```
sortterms_ : seq Term ⟶ seq Term
─────────────────────────────────
∀ t : seq Term •
    sortterms t ⇔ ∀ i, j : 1..#t | i < j •
         (t i).score ≠ (t j).score ⇒
              (t i).score > (t j).score
         (t i).score = (t j).score ⇒
              (t i).position < (t j).position
```

The system stores the return value of the function *detectstems*, *scoreterms* and *sortterms* in the global variable *terms*.

```
┌──── Extract ──────────────────────────────
...
──────────────────────────
terms = sortterms(scoreterms(detectstems(words)))
...
└─────────────────────────────────────────────
```

7.2.7 Selecting Phrases

Step 3 involves the selection of all the noun phrases in the document. This is because almost all the keyphrases are noun phrases and they normally contain less than four words and match the following pattern [108]:

$$(\text{NN} \mid \text{NNS} \mid \text{NNP} \mid \text{NNPS} \mid \text{JJ})^{0..2}\ (\text{NN} \mid \text{NNS} \mid \text{NNP} \mid \text{NNPS} \mid \text{VBG})$$

This pattern means zero, one or two nouns or adjectives (NN | NNS | NNP | NNPS | JJ) followed by a noun or a gerund (NN | NNS | NNP | NNPS | VBG).

```
selectphrases_ : seq Tag ⟶ seq String
─────────────────────────────
∀ t : seq Tag •
    selectphrases  t =
         squash { i : 1..#t |
              (t i).p ∈ { nn, nns, nnp, nnps, vbg } •
              i ⟼ (t i).s }
         ⌢
         squash { i : 2..#t |
              (t i-1).p ∈ { nn, nns, nnp, nnps, jj } ∧
              (t i).p ∈ { nn, nns, nnp, nnps, vbg } •
              i ⟼ ((t i-1).s concat (t i).s) }
         ⌢
         squash { i : 3..#t |
              (t i-2).p ∈ { nn, nns, nnp, nnps, jj } ∧
              (t i-1).p ∈ { nn, nns, nnp, nnps, jj } ∧
              (t i).p ∈ { nn, nns, nnp, nnps, vbg } •
              i ⟼ ((t i-2).s concat (t i-1).s
                   concat (t i).s) }
```

The system tags the input document before passing it to the function *selectphrases*, and stores the return value of this function in the global variable *phrases*.

```
┌──── Extract ──────────────────────────────
raw_document? : String
...
──────────────────────────
phrases = selectphrases(tag(raw_document?))
...
└──────────────────────────────────────────
```

7.2.8 Scoring Term Phrases

Similar to Step 2, Step 4 involves six closely related tasks:

1. Stem the selected phrases

2. Detect equivalent stem phrases

3. Delete term phrases that occur only once in the document

4. Calculate the *TF×IDF* (using the standard TF and Kea's IDF), *position*, *title*, and *number of terms* of each term phrase

5. Assign a score to each term phrase based on these attributes

6. Sort the term phrases in descending order of score (if two term phrases have the same score, they are ranked in ascending order of *position* followed by descending order of *number of terms*)

Similar to Step 2, the function *stems* performs Task 1, and *detectstems* performs Task 2 and 3. For details of these functions, see Section 7.2.6. The remaining tasks are performed by two functions: the function *scoretermphrases* performs Task 4 and 5, and *sorttermphrases* performs Task 6.

For each term phrase, the value of *TF×IDF*, *position*, *title*, and *number of terms* is calculated. Informal definitions and descriptions of these attributes can be found in Section 1.3. Each term phrase is then scored based on these attributes.

```
scoretermphrases_ : seq String ⟶ seq TermPhrase
─────────────────────────────────────────────────
∀ s : seq String •
    scoretermphrases s ⇔ ∀ t : seq TermPhrase •
          ...
     ∀ i : 1..#t •
          ...
    (t i).tf =
          #((phrases ⨾ stemmed_phrases) ⊳ { (t i).str })
          ...
    (t i).no_of_terms =
          #tokenize((t i).str)
    (t i).score =
          scoretermphrase((t i).tf*(t i).idf,
          (t i).position, (t i).title,
          (t i).no_of_terms)
```

The function *sorttermphrases* sorts the objects in the sequence by score followed by *position* and *no_of_terms*. If object *a* has a higher score than object *b*, *a* is nearer the top of the sequence; if *a* has the same score as *b*, the object with a smaller *position* value is nearer the top; if *a* has the same score and *position* value as *b*, the object with a larger *no_of_terms* value is nearer the top.

```
sorttermphrases_ : seq TermPhrase ⟶ seq TermPhrase
─────────────────────────────────
∀ t : seq TermPhrase •
    sorttermphrases t ⇔ ∀ i, j : 1..#t | i < j •
         (t i).score ≠ (t j).score ⇒
              (t i).score > (t j).score
         (t i).score = (t j).score ∧
         (t i).position ≠ (t j).position ⇒
              (t i).position < (t j).position
         (t i).score = (t j).score ∧
         (t i).position = (t j).position ⇒
              (t i).no_of_terms > (t j).no_of_terms
```

The system uses the function *stems* to stem the selected phrases and stores the return value of this function in the global variable *stemmed_phrases*. It then stores the return value of the function *detectstems*, *scoretermphrases* and *sorttermphrases* in the global variable *term_phrases*.

```
┌──── Extract ──────────────────────────────
...
─────────────────────────
stemmed_phrases = stems(phrases)
term_phrases = sorttermphrases(scoretermphrases
    (detectstems(phrases)))
...
└──────────────────────────────────────────
```

7.2.9 Expanding Terms

Step 5 involves expanding single terms to term phrases. For each term, find all the term phrases that contain the term, and link it with the highest scoring term phrase. The result is a list of term phrases ordered by the scores calculated in Step 2.

Suppose that the term phrase ‘integer fact algorithm’ and ‘fact’ appear in the first and second position of the sequence *p* respectively. The term ‘fact’ (stem of ‘factorization’) will link to ‘integer fact algorithm’ instead of ‘fact’ (see Figure 7.2).

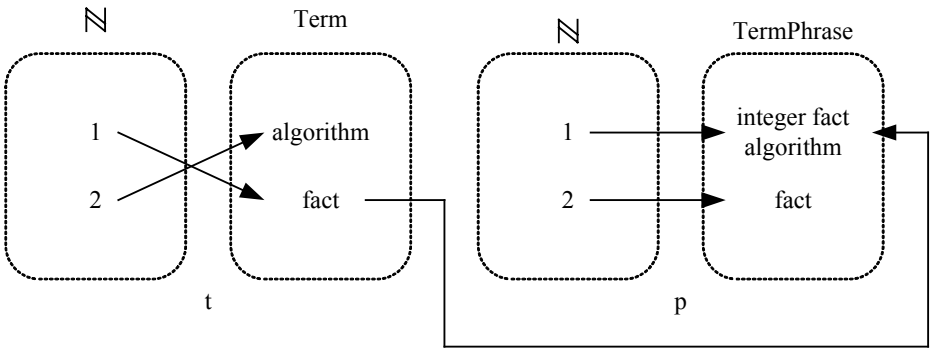

Figure 7.2: Expanding terms to term phrases

The function *expandterms* ensures that no term links to more than one term phrase and uses the scores calculated in Step 2 to rank the output sequence.

```
expandterms_ _ : (seq Term × seq TermPhrase)
    ⟶ seq TermPhrase
─────────────────────────────────
∀ t : seq Term; p : seq TermPhrase •
    expandterms (t, p) =
         squash { i : 1..#t •
         i ⟼ (p (min { j : 1..#p |
         ((p j).str indexof (t i).str) ≠ 0 • j } ))
         }
```

7.2.10 Dropping Duplicates

Step 6 involves the elimination of duplicates from the list of term phrases. More than one term may link to the same term phrase (i.e. there may be converging arrows in the graph). If that is the case, the term phrase will be linked to the highest scoring term.

Suppose that the terms ‘fact’ (appears in the first position) and ‘algorithm’ (appears in the second position) are expanded to the term phrase ‘integer fact algorithm’. ‘Fact’ instead of ‘algorithm’ will link to ‘integer fact algorithm’ (see Figure 7.3).

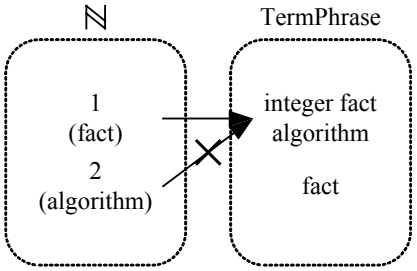

Figure 7.3: Deleting duplicate term phrases

The function *dropduplicates* ensures that no term phrase appears more than once in the output sequence.

```
dropduplicates_ : seq TermPhrase ⟶ iseq String
──────────────────────────────
∀ t : seq TermPhrase •
    dropduplicates t =
         squash { i : ran t •
              min(dom(t ⊳ { i })) ⟼ i.str }
```

The system stores the return value of the function *expandterms* and *dropduplicates* in the global variable *key_term_phrases*.

```
┌──── Extract ──────────────────────────────
...
──────────────────────────
key_term_phrases =
    dropduplicates(expandterms(terms, term_phrases))
...
└──────────────────────────────────────────
```

7.2.11 Displaying Output

Step 7 involves two closely related tasks:

1. Identify the most frequent corresponding phrase in the input document for each of the term phrases. If a term phrase is linked to more than one phrase, the most frequent phrase will be chosen.

2. Delete subphrases if they do not perform better than their superphrases. If phrase P_1 occurs within phrase P_2, P_1 is a subphrase of P_2 and P_2 is a superphrase of P_1. If a phrase is a subphrase of another phrase, it will only be accepted as a keyphrase if it is ranked higher; otherwise it will be deleted from the output list.

The function *displayoutput* performs Task 1, and *postprocess* performs Task 2. These functions will be discussed in this section.

Suppose that the term phrase 'mathemat' is linked to 'mathematics' (appears twice in the input document) and 'mathematician' (appears once). 'Mathematics' instead of 'mathematician' will be chosen for output (see Figure 7.4).

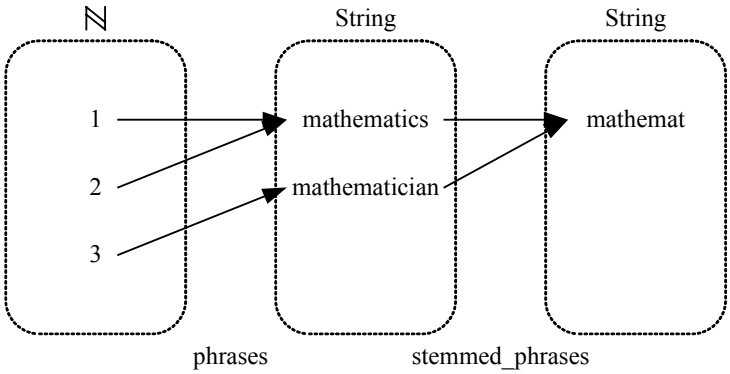

Figure 7.4: Identifying the most frequent phrases

The function *displayoutput* takes a sequence of term phrases as input and returns a sequence of phrases. The precondition of *displayoutput* is that `ran is ⊆ ran stemmed_phrases`.

```
displayoutput_ : iseq String ⟶ iseq String
─────────────────────────────────────────
∀ is : iseq String •
    displayoutput is = is ⨾
    { i : ran is •
    i ⟼ { S : dom(stemmed_phrases ⊳ { i });
         m : String | m ∈ S  ∧
         (∀ n : S •
         #(phrases ⊳ { m }) ≥ #(phrases ⊳ { n })) • m }
    }
```

Suppose that the phrase ‘integer factorization algorithm’ appears in the first position of the output list and ‘factorization’ appears in the second position, ‘factorization’ will be removed because it is a subphrase of ‘integer factorization algorithm’ and is ranked lower (see Figure 7.5).

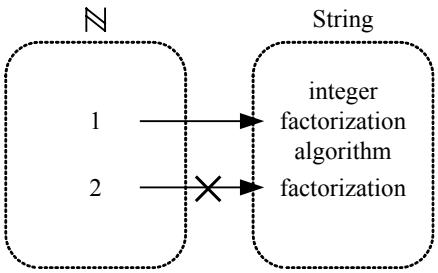

Figure 7.5: Deleting inferior subphrases

```
postprocess_ : iseq String ⟶ iseq String
─────────────────────────────────────────
∀ is : iseq String •
    postprocess is =
         squash { j : 1..#is |
         (∀ i : 1..#is • i < j ∧
         ((is i) indexof (is j)) = 0) • j ⟼ (is j)}
```

The system stores the return value of the function *displayoutput* and *postprocess* in the keyphrase list *keyphrases’*.

```
┌──── Extract ──────────────────────────────────
│...
├──────────────────────────
│keyphrases' =
│    postprocess(displayoutput(key_term_phrases))
│...
└──────────────────────────────────────────────
```

7.2.12 Extracting Keyphrases

The keyphrase extraction system contains a list of keyphrases. A phrase which is nearer the top of the list is more likely to be a keyphrase. No keyphrase should appear in the list more than once (this is reinforced by Step 6), and all the keyphrases should contain less than four words (this is reinforced by Step 3).

```
┌──── KeyphraseExtraction ──────────────────────
│keyphrases : iseq String
├──────────────────────────
│∀ s : ran keyphrases • #(tokenize(s)) < 4
└──────────────────────────────────────────────
```

When the system is initialized, the keyphrase list is empty.

```
┌──── KeyphraseExtractionInit ──────────────────
│KeyphraseExtraction'
├──────────────────────────
│keyphrases' = ∅
└──────────────────────────────────────────────
```

To describe an operation which may change the contents of the keyphrase list, we include two copies of the system state:

```
┌──── ΔKeyphraseExtraction ──────────────────────
│KeyphraseExtraction
│KeyphraseExtraction'
└──────────────────────────────────────────────
```

When the user provides the system with a document *raw_document?*, the title of this document *raw_title?*, and the desired number of output keyphrases *num?*, the system stems the title, tags and stems the document, goes through all the steps mentioned above, and displays a list of the top *num?* keyphrases.

```
┌──── Extract ──────────────────────────────
│ΔKeyphraseExtraction
│raw_document? : String
│raw_title? : String
│num? : ℕ
│out! : iseq String
├──────────────────────
│document = stem(raw_document?)
│title = stem(raw_title?)
│proper_nouns = getpropernouns(tag(raw_document?))
│words = selectwords(tag(raw_document?))
│stemmed_words = stems(words)
│terms = sortterms(scoreterms(detectstems(words)))
│phrases = selectphrases(tag(raw_document?))
│stemmed_phrases = stems(phrases)
│term_phrases = sorttermphrases(scoretermphrases
│    (detectstems(phrases)))
│key_term_phrases =
│    dropduplicates(expandterms(terms, term_phrases))
│keyphrases' =
│    postprocess(displayoutput(key_term_phrases))
│out! = { i : 1..num? | #keyphrases' ≥ num? •
│              i ⟼ keyphrases' i }
└──────────────────────────────────────────
```

7.3 Summary

This appendix describes the KE algorithm using the Z notation. The algorithm has been specified in Z and explained with examples and informal English descriptions.

References

1. James Allen, Natural Language Understanding, Benjamin/ Cummings, 2nd Edition, 1995

2. Ricardo Baeza-Yates and Berthier Ribeiro-Neto, Modern Information Retrieval, ACM Press, 1999

3. Alvaro Barbosa, Overview of Text Summarization in the Context of Information Retrieval and Interpretation: Applications for Web Pages Summarization, Research Article for the Audiovisual Institute of the Pompeu Fabra University in the Context of the Doctorate Program in Computer Science and Digital Communication, 2001

4. Peter Biebricher, Norbert Fuhr, Gerhard Knorz, Gerhard Lustig and Michael Schwantner, The Automatic Indexing System AIR/ PHYS – from Research to Application, Proceedings of 11th International Conference on Research and Development in Information Retrieval, Grenoble, France, ACM Press, pp. 333-342, 1988

5. Joseph Bigus, Data Mining with Neural Networks, McGraw-Hill, 1996

6. Christopher Bishop, Neural Networks for Pattern Recognition, Oxford University Press, 1995

7. Ronald Brandow, Karl Mitze and Lisa Rau, Automatic Condensation of Electronic Publications by Sentence Selection, Information Processing and Management, Vol. 31, No. 5, pp. 675-685, 1995

8. Eric Brill, A Corpus-Based Approach to Language Learning, Ph.D. Dissertation, Department of Computer and Information Science, University of Pennsylvania, USA, 1993

9. Eric Brill, A Simple Rule-Based Part of Speech Tagger, Proceedings of 3rd Conference on Applied Natural Language Processing, Trento, Italy, ACM Press, pp. 152-155, 1992

10. Sergey Brin and Lawrence Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, Proceedings of 7th International Conference on World Wide Web 7, Brisbane, Australia, Elsevier, pp. 107-117, 1998

11. British National Corpus, http://www.natcorp.ox.ac.uk/

12. John Broglio, James Callan and W. Bruce Croft, INQUERY System Overview, Proceedings of Tipster Text Program: Phase 1, Virginia, USA, Association for Computational Linguistics, pp. 47-67, 1993

13. John Bullinaria, John Bullinaria's Step by Step Guide to Implementing a Neural Network in C, http://www.cs.bham.ac.uk/~jxb/NN/nn.html

14. Soumen Chakrabarti, Byron Dom, David Gibson, Jon Kleinberg, Ravi Kumar, Prabhakar Raghavan, Sridhar Rajagopalan and Andrew Tomkins, Mining the Link Structure of the World Wide Web, IEEE Computer, Vol. 32, No. 8, pp. 60-67, 1999

15. Soumen Chakrabarti, Byron Dom, Prabhakar Raghavan, Sridhar Rajagopalan, David Gibson and Jon Kleinberg, Automatic Resource List Compilation by Analyzing Hyperlink Structure and Associated Text, Proceedings of 7th International World Wide Web Conference, Brisbane, Australia, Elsevier, pp. 65-74, 1998

16. Hsinchun Chen, Machine Learning for Information Retrieval: Neural Networks, Symbolic Learning, and Genetic Algorithms, Journal of the American Society for Information Science, Vol. 46, Issue 3, pp. 194-216, 1995

17. Mo Chen, Jian-Tao Sun, Hua-Jun Zeng and Kwok-Yan Lam, A Practical System of Keyphrase Extraction for Web Pages, Proceedings of 14th ACM International Conference on Information and Knowledge Management, Bremen, Germany, ACM Press, pp. 277-278, 2005

18. Jim Cowie and Wendy Lehnert, Information Extraction, Communications of the ACM, Vol. 39, No. 1, pp. 80-91, 1996

19. Jim Cowie and Yorick Wilks, Information Extraction, http://www.dcs.shef.ac.uk/~yorick/papers/infoext.pdf

20. W. Bruce Croft and David Harper, Using Probabilistic Models of Document Retrieval without Relevance Information, Journal of Documentation, Vol. 35, No. 4, pp. 285-295, 1979

21. Hamish Cunningham, Information Extraction – A User Guide (2nd Edition), http://www.dcs.shef.ac.uk/~hamish/IE/userguide/main.html

22. Hamish Cunningham, Information Extraction, http://gate.ac.uk/ie/

23. Sally Cunningham, Geoffrey Holmes, Jamie Littin, Russell Beale and Ian Witten, Applying Connectionist Models to Information Retrieval, Brain-Like Computing and Intelligent Information Systems, Springer-Verlag, pp. 435-457, 1997

24. Sally Cunningham, James Littin and Ian Witten, Applications of Machine Learning in Information Retrieval, Technical Report 97/ 6, Department of Computer Science, University of Waikato, 1997

25. Bo Curry and David Rumelhart, MSnet: A Neural Network that Classifies Mass Spectra, HP Labs Technical Report HPL-90-161, 1990

26. Ernesto D'Avanzo and Bernardo Magnini, A Keyphrase-Based Approach to Summarization: The LAKE System at DUC-2005, Document Understanding Workshop, Vancouver, Canada, 2005

27. Ernesto D'Avanzo, Bernardo Magnini and Alessandro Vallin, Keyphrase Extraction for Summarization Purposes: The LAKE System at DUC-2004, Document Understanding Workshop, Boston, USA, 2004

28. H. Edmundson, New Methods in Automatic Extracting, Journal of the ACM, Vol. 16, Issue 2, pp. 264-285, 1969

29. Lauene Fausett, Fundamentals of Neural Networks: Architectures, Algorithms, and Applications, Prentice-Hall, 1994

30. Usama Fayyad and Keki Irani, Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning, Proceedings of 13th International Joint Conference on Artificial Intelligence, Chambery, France, Morgan Kaufmann, pp. 1022-1027, 1993

31. Rand Fishkin and Jeff Pollard, Search Engine Ranking Factors V2, http://www.seomoz.org/article/search-ranking-factors

32. William Frakes and Ricardo Baeza-Yates, Information Retrieval: Data Structures and Algorithms, Prentice Hall, 1992

33. W. Nelson Francis and Henry Kucera, Brown Corpus Manual, http://helmer.aksis.uib.no/icame/brown/bcm.html

34. Eibe Frank, Gordon Paynter, Ian Witten, Carl Gutwin and Craig Nevill-Manning, Domain-Specific Keyphrase Extraction, Proceedings of 16th International Joint Conference on Artificial Intelligence (IJCAI-99), California, USA, Morgan Kaufmann, pp. 668-673, 1999

35. David Gibson, Jon Kleinberg and Prabhakar Raghavan, Inferring Web Communities from Link Topology, Proceedings of 9th ACM Conference on Hypertext and Hypermedia, Pennsylvania, USA, ACM Press, pp. 225-234, 1998

36. Jade Goldstein, Mark Kantrowitz, Vibhu Mittal and Jaime Carbonel, Summarizing Text Documents: Sentence Selection and Evaluation Metrics, Proceedings of 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-99), California, USA, ACM Press, pp. 121-128, 1999

37. Ralph Grishman and Beth Sundheim, Message Understanding Conference – 6: A Brief History, Proceedings of 16th International Conference on Computational Linguistics, Copenhagen, Denmark, Association for Computational Linguistics, pp. 466-471, 1996

38. Kevin Gurney, An Introduction to Neural Networks, UCL Press, 1997

39. Carl Gutwin, Gordon Paynter, Ian Witten, Craig Nevill-Manning and Eibe Frank, Improving Browsing in Digital Libraries with Keyphrase Indexes, Journal of Decision Support Systems, Vol. 27, Issue 1-2, pp. 81-104, 1999

40. Jiawei Han and Micheline Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, 2001

41. Donna Harman, Overview of the Third Text Retrieval Conference (TREC-3), Proceedings of 3rd Text Retrieval Conference (TREC-3), Maryland, USA, NIST Special Publication 500-225, 1994

42. Donna Harman, Relevance Feedback Revisited, Proceedings of 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Copenhagen, Denmark, ACM Press, pp. 1-10, 1992

43. Robert Hecht-Nielsen, Neurocomputing, Addison-Wesley, 1990

44. Charles Hildreth, Online Library Catalogues as Information Retrieval Systems: What Can We Learn From Research?, Proceedings of the Silver Jubilee Conference of the City University's Department of Information Science, London, UK, Taylor Graham, pp. 9-25, 1988

45. Rob Hooper and Chris Paice, The Lancaster Stemming Algorithm, http://www.comp.lancs.ac.uk/computing/research/stemming/index.htm

46. David Hull, Stemming Algorithms: A Case Study for Detailed Evaluation, Journal of the American Society of Information Science, Vol. 47, No. 1, pp.70-84, 1996

47. Jonathan Jacky, The Way of Z: Practical Programming with Formal Methods, Cambridge University Press, 1997

48. Taeho Jo, Neural Based Approach to Keyword Extraction from Documents, Proceedings of 2003 International Conference on Computational Science and its Applications (ICCSA 2003), Montreal, Canada, Springer-Verlag, pp. 456-461, 2003

49. Frances Johnson, Chris Paice, William Black, and A. Neal, The Application of Linguistic Processing to Automatic Abstract Generation, Journal of Document and Text Management, Vol. 1, No. 3, pp. 215-241, 1993

50. Steve Jones and Gordon Paynter, Automatic Extraction of Document Keyphrases for Use in Digital Libraries: Evaluation and Applications, Journal of the American Society for Information Science and Technology, Vol. 53, Issue 8, pp. 653-677, 2002

51. Steve Jones and Gordon Paynter, Human Evaluation of Kea, An Automatic Keyphrasing System, Proceedings of 1st ACM/ IEEE-CS Joint Conference on Digital Libraries, Virginia, USA, ACM Press, pp. 148-156, 2001

52. Steve Jones and Mark Staveley, Phrasier: A System for Interactive Document Retrieval using Keyphrases, Proceedings of 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, California, United States, ACM Press, pp. 160-167, 1999

53. Jon Kleinberg, Authoritative Sources in a Hyperlinked Environment, Journal of the ACM, Vol. 46, No. 5, pp. 604-632, 1999

54. Kevin Knight, Connectionist Ideas and Algorithms, Communications of the ACM, Vol. 33, No. 11, pp. 58-74, 1990

55. Raymond Kosala and Hendrik Blockeel, Web Mining Research: A Survey, SIGKDD Explorations: Newsletter of the Special Interest Group (SIG) on Knowledge Discovery & Data Mining, ACM, Vol. 2, Issue 1, pp. 1-15, 2000

56. Robert Krovetz, Viewing Morphology as an Inference Process, Proceedings of 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Pennsylvania, USA, ACM Press, pp. 191-202, 1993

57. Bruce Krulwich and Chad Burkey, Learning User Information Interests through the Extraction of Semantically Significant Phrases, Proceedings of AAAI Spring Symposium on Machine Learning in Information Access, California, USA, AAAI Press, pp. 110-112, 1996

58. Julian Kupiec, Jan Pedersen and Francine Chen, A Trainable Document Summarizer, Proceedings of 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR-95), Washington, USA, ACM Press, pp. 68-73, 1995

59. Savio Lam and Dik Lee, Feature Reduction for Neural Network Based Text Categorization, Proceedings of 6th International Conference on Database Systems for Advanced Applications (DASFAA-99), Taiwan, ROC, IEEE Computer, pp. 195-202, 1999

60. Ray Larson, Jerome McDonough, Paul O'Leary, Lucy Kuntz and Ralph Moon, Cheshire II: Designing a Next-Generation Online Catalog, Journal of the American Society for Information Science, Vol. 47, Issue 7, pp. 555-567, 1996

61. Gordon Linoff and Michael Berry, Mining the Web: Transforming Customer Data into Customer Value, Wiley, 2002

62. Bing Liu and Kevin Chen-Chuan-Chang, Editorial: Special Issue on Web Content Mining, ACM SIGKDD Explorations Newsletter, Vol. 6, Issue 2, pp. 1-4, 2004

63. Julie Lovins, Development of a Stemming Algorithm, Mechanical Translation and Computational Linguistics, Vol. 11, No. 1, pp. 22-31. 1968

64. Hans Luhn, The Automatic Derivation of Information Retrieval Encodements from Machine-Readable Texts, Information Retrieval and Machine Translation, Vol. 3, Part 2, pp. 1021-1028, New York Interscience Publication, 1961

65. Hans Luhn, The Automatic Creation of Literature Abstracts, IBM Journal of Research and Development, Vol. 2, No. 2, pp. 159-165, 1958

66. Yuan Lui, Extraction of Significant Phrases from Text, International Journal of Computer Science, Vol. 2, No. 2, pp. 101-109, 2007

67. Yuan Lui, Learning to Extract Significant Phrases from Text, Research Report RR-07-01, Oxford University Computing Laboratory, UK, 2007

68. Yuan Lui, An Improved Keyphrase Extraction Algorithm, Proceedings of PREP2005, Lancaster, UK, 2005

69. Yuan Lui, Richard Brent and Ani Calinescu, Extracting Significant Phrases from Text, Proceedings of IEEE Data Mining and Information Retrieval, Ontario, Canada, IEEE Computer, pp. 361-366, 2007

70. Inderjeet Mani, Automatic Summarization, John Benjamins, 2001

71. Olena Medelyan and Ian Witten, Thesaurus Based Automatic Keyphrase Indexing, Proceedings of 6th ACM/ IEEE-CS Joint Conference on Digital Libraries, North Carolina, USA, ACM Press, pp. 296-297, 2006

72. Tom Mitchell, Machine Learning, McGraw-Hill, 1997

73. Chuck Musciano and Bill Kennedy, HTML: The Definitive Guide, 3rd Edition, O'Reilly, 1998

74. Mary Okurowski, Information Extraction Overview, Proceedings of Tipster Text Program: Phase 1, Virginia, USA, Association for Computational Linguistics, pp. 117-121, 1993

75. J. Scott Olsson, Douglas Oard and Jan Hajic, Cross-Language Text Classification, Proceedings of 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Salvador, Brazil, ACM Press, pp. 645-646, 2005

76. Chris Paice, The Automatic Generation of Literature Abstracts: An Approach Based on the Identification of Self-Indicating Phrases, Proceedings of 3rd Annual ACM Conference on Research and Development in Information Retrieval, Cambridge, UK, Butterworth, pp. 172-191, 1980

77. Paper Collection, http://www.usc.cuhk.edu.hk/wk.asp

78. Martin Porter, An Algorithm for Suffix Stripping, Program, Vol. 14, No. 3, pp. 130-137, 1980

79. Martin Porter and Richard Boulton, The Lovins Stemming Algorithm, http://snowball.tartarus.org/algorithms/lovins/stemmer.html

80. Lutz Prechelt, Neural Networks FAQ, http://www.cs.cmu.edu/Groups/AI/html/faqs/ai/neural/faq.html

81. Dorian Pyle, Data Preparation for Data Mining, Morgan Kaufmann, 1999

82. J. Ross Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, 1993

83. Brian Ripley, Pattern Recognition and Neural Networks, Cambridge University Press, 1996

84. Alexander Robertson and Robert Gaizauskas, On the Marriage of Information Retrieval and Information Extraction, Proceedings of 19th Annual BCS-IRSG Colloquium on IR Research, Aberdeen, UK, Springer-Verlag, pp.1-8, 1997

85. Stephen Robertson and Karen Sparck-Jones, Simple, Proven Approaches to Text Retrieval, Technical Report TR356, Cambridge University Computer Laboratory, UK, 1997

86. David Rumelhart, Bernard Widrow and Michael Lehr, The Basic Ideas in Neural Networks, Communications of the ACM, Vol. 37, No. 3, pp. 87-92, 1994

87. Gerard Salton and Christopher Buckley, Improving Retrieval Performance by Relevance Feedback, Journal of the American Society for Information Science, Vol. 41, Issue 4, pp. 288-297, 1990

88. Gerard Salton and Christopher Buckley, Term-Weighting Approaches in Automatic Text Retrieval, Information Processing and Management, Vol. 24, No. 5, pp. 513-523, 1988

89. Gerald Salton and Michael McGill, The SMART and SIRE Experimental Retrieval Systems, Introduction to Modern Information Retrieval, McGraw-Hill, pp. 118-156, 1983

90. Gerard Salton and Michael McGill, Introduction to Modern Information Retrieval, McGraw-Hill, 1983

91. Serge Sharoff, Chinese Tokenisation and Tagging. http://corpus.leeds.ac.uk/tools/zh/

92. Liddy Shriver and Christopher Small, The SearchLight Proxy Server, Bell Laboratories, http://www.bell-labs.com/project/searchlight/

93. Min Song, Il-Yeol Song and Xiaohua Hu, KPSpotter: A Flexible Information Gain-based Keyphrase Extraction System, Proceedings of 5th ACM International Workshop on Web Information and Data Management, Louisiana, USA, ACM Press, pp. 50-53, 2003

94. Karen Sparck-Jones, Language Modelling's Generative Model: Is It Rational?, http://www.cl.cam.ac.uk/~ksj/langmodnote4.pdf

95. Karen Sparck-Jones and Peter Willett, Readings in Information Retrieval, Morgan Kaufmann, 1997

96. Mike Spivey, The Z Notation: A Reference Manual, Prentice-Hall, 1989

97. Danny Sullivan, Death of a Meta Tag, http://searchenginewatch.com/showPage.html?page=2165061

98. Beth Sundheim, Overview of Results of the MUC-6 Evaluation, Proceedings of 6th Message Understanding Conference (MUC-6), Maryland, USA, Association for Computational Linguistics, pp. 13-31, 1995

99. Beth Sundheim and Nancy Chinchor, Survey of the Message Understanding Conferences, Proceedings of the Workshop on Human Language Technology, New Jersey, USA, Association for Computational Linguistics, pp. 56-60, 1993

100. Chia-Hung Tai, Chinese Word Segmentation System with Unknown Word Identification, http://godel.iis.sinica.edu.tw/CKIP/engversion/wordsegment.htm

101. Pang-Ning Tan, Michael Steinbach and Vipin Kumar, Introduction to Data Mining, Addison-Wesley, 2006

102. Alberto Tellez-Valero, Manuel Montes-y-Gomez and Luis Villasenor-Pineda, A Machine Learning Approach to Information Extraction, Proceedings of 6th International Conference on Intelligent Text Processing and Computational Linguistics (CICLing-2005), Mexico City, Mexico, Springer-Verlag, pp. 539-547, 2005

103. Text Retrieval Conferences (TREC), http://trec.nist.gov/

104. Daniel Tkach, Text Mining Technology: Turning Information into Knowledge, A White Paper from IBM, IBM Software Solutions, 1998

105. Peter Turney, Coherent Keyphrase Extraction via Web Mining, Proceedings of 18th International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico, CogPrints, pp. 434-439, 2003

106. Peter Turney, Learning Algorithms for Keyphrase Extraction, Information Retrieval, Vol. 2, No. 4, pp. 303-336, 2000

107. Peter Turney, Learning to Extract Keyphrases from Text, Technical Report ERB-1057, Institute for Information Technology, National Research Council, Canada, 1999

108. Peter Turney, Extraction of Keyphrases from Text: Evaluation of Four Algorithms, Technical Report ERB-1051, Institute for Information Technology, National Research Council, Canada, 1997

109. Kees van Deemter and Rodger Kibble, On Coreferring: Coreference in MUC and Related Annotation Schemes, Computational Linguistics, Vol. 26, Issue 4, pp. 629-637, 2000

110. Keith van Rijsbergen, Information Retrieval, http://www.dcs.gla.ac.uk/Keith/Preface.html

111. Ellen Voorhees, Overview of TREC 2004, Proceedings of 13th Text Retrieval Conference (TREC 2004), Maryland, USA, NIST Special Publication SP 500-261, 2004

112. Ryen White, Implicit Feedback for Interactive Information Retrieval, Ph.D. Dissertation, Department of Computing Science, University of Glasgow, UK, 2004

113. Ross Wilkinson and Philip Hingston, Using the Cosine Measure in a Neural Network for Document Retrieval, Proceedings of 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Illinois, USA, ACM Press, pp. 202-210, 1991

114. Ian Witten and Eibe Frank, Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations, Morgan Kaufmann, 2000

115. Ian Witten, Alistair Moffat and Timothy Bell, Managing Gigabytes: Compressing and Indexing Documents and Images, 2nd Edition, Morgan Kaufmann, 1999

116. Jim Woodcock and Jim Davis, Using Z: Specification, Refinement, and Proof, Prentice-Hall, 1996

117. Jim Woodcock and Martin Loomes, Software Engineering Mathematics, Pitman, 1988

118. Yi-fang Wu, Quanzhi Li, Razvan Bot and Xin Chen, Domain-Specific Keyphrase Extraction, Proceedings of 14th ACM International Conference on Information and Knowledge Management, Bremen, Germany, ACM Press, pp. 283-284, 2005

119. Lan You, Yongping Du, Jiayin Ge, Xuanjing Huang and Lide Wu, BBS Based Hot Topic Retrieval Using Back-Propagation Neural Network, Proceedings of 1st International Joint Conference on Natural Language Processing (IJCNLP 2004), Hainan Island, China, Springer-Verlag, pp. 139-148, 2005

120. Osmar Zaiane, Jiawei Han, Ze-Nian Li, Sonny Chee and Jenny Chiang, MultiMediaMiner: A System Prototype for MultiMedia Data Mining, Proceedings of 1998 ACM SIGMOD International Conference on Management of Data, Washington, USA, ACM Press, pp. 581-583, 1998

121. A. Zanasi, Text Mining and its Applications, WIT Press, 2005

122. Kevin Zhang, The ICTCLAS System, http://www.nlp.org.cn/project/project.php?proj_id=6

123. Yongzheng Zhang, Nur Zincir-Heywood and Evangelos Milios, World Wide Web Site Summarization, Web Intelligence and Agent Systems, Vol. 2, Issue 1, pp. 39-53, 2004

124. George Zipf, Human Behaviour and the Principle of Least Effort, Addison-Wesley, 1949