Using Semantics in Document Representation: A Lexical Chain Approach

A THESIS

submitted by

DINAKAR JAYARAJAN

for the award of the degree

of

MASTER OF SCIENCE

(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS

June 2009

“The future belongs to those who believe in the beauty of their dreams”

- Eleanor Roosevelt

Dedicated to

My Parents

THESIS CERTIFICATE

This is to certify that the thesis entitled **Using Semantics in Document Representation: A Lexical Chain Approach** submitted by **Dinakar Jayarajan**, to the Indian Institute of Technology Madras for the award of the degree of Master of Science (by research), is a bona-fide record of the research work carried out by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Date:

Chennai 600 036. (Dr. B. Ravindran) (Dr. Dipti Deodhare)

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my research advisors Dr. B Ravindran and Dr. Dipti Deodhare, Sc 'F' for providing me excellent motivation and support throughout my M.S. Program. Their dedication towards research, desire for perfection and utmost concern inspired me to put in my best efforts for research work. Both of them have been major pillars of support for me over the last many years.

My stint at IIT Madras has also given me many new friends and I am sure that these new friends will last a life time. I would also like to thank my colleagues at CAIR, Bangalore for supporting me during this period. I must specifically acknowledge my previous director, Shri N Sitaram and my HR Director, Shri M R Keshorey for allowing me to pursue the MS program at IIT Madras.

I would also like to thank my parents who contantly kept reminding me that I should enroll for and earn a Masters degree. My brother Jayanth and my cousin Rajesh have been very helpful in arranging for my accomodation, transport and entertainment whenever I visited Chennai. Words are not enough to express my gratitdue to my wife Ashy, who entered my life roughly around the time I started to write this thesis. She patiently bore the brunt of me writing this thesis!

Dinakar Jayarajan

ABSTRACT

Automatic classification and clustering are two of the most common operations performed on text documents. Numerous algorithms have been proposed for this and invariably, all of these algorithms use some variation of the vector space model to represent the documents. Traditionally, the Bag of Words (BoW) representation is used to model the documents in a vector space. The BoW scheme is a simple and popular scheme, but it suffers from numerous drawbacks. The chief among them is that the feature vectors generated using BoW results in very large dimensional vectors. This creates problems with most machine learning algorithms where the high dimensionality severely affects the performance. This fact is manifested in the current thinking in the machine learning community, that some sort of a dimensionality reduction is a beneficial preprocessing step for applying machine learning algorithms on textual data. BoW also fails to capture the semantics contained in the documents properly.

In this thesis, we present a new representation for documents based on lexical chains. This representation addresses both the problems with BoW - it achieves a significant reduction in the dimensionality and captures some of the semantics present in the data. We present an improved algorithm to compute lexical chains and generate feature vectors using these chains.

We evaluate our approach using datasets derived from the 20 Newsgroups corpus. This corpus is a collection of 18,941 documents across 20 Usenet groups related to topics such as computers, sports, vehicles, religion, *etc*. We compare the performance of the lexical chain based document features against the BoW features on two machine learning tasks - clustering and classification. We also present a preliminary algorithm for soft clustering using the lexical chains based representation, to facilitate topic detection from a document corpus.

CONTENTS

LIST OF TABLES

List of Algorithms

LIST OF FIGURES

CHAPTER 1

Introduction

Humans have an innate ability to learn, generalise and discriminate between objects which exist in this universe. For example, we were not taught an equation or given a formula for recognising the letters of the alphabet. But, by seeing many examples and variations of the alphabets over a period of time, we are able to learn them and subsequently recognise large variations in the alphabets very effortlessly. Even though computers have a long way to go before they can truly mimic humans, they are slowly inching towards this goal.

Machine learning is a branch of artificial intelligence which studies mechanisms to mimic the ability of humans to learn. Machine learning strives to get the computer to learn tasks such as discriminating between objects, segregating similar objects from dissimilar ones and learning from experience. These tasks are formally known as supervised, unsupervised and reinforcement learning in the machine learning parlance.

In supervised learning, the system is presented with a set of data which is labelled into various categories and involves learning a function which maps the data to the categories. This function is then used to map an unseen instance of the data to its corresponding category. Unsupervised learning on the other hand works on unlabelled data and involves grouping this data based on its characteristics, *i.e.*, infer potential categories from unlabelled data. Reinforcement learning is a system which learns an effective way of doing a task from the experience of doing the task and feedback from the environment on the outcome.

The proliferation of digital data gathering and storing mechanisms has resulted in humongous volumes of data being generated. This data has to be processed to derive the information

content latent in them. It is impossible to process such large volumes of data manually. Naturally the demand for machine learning solutions to process this data is rising. Automatic processing on a large scale is not just a feasible solution, but also the only practical way of making sense of all this data.

Document classification and clustering are two common operations performed on text documents. Document classification is an instance of supervised learning. Here the goal is to automatically classify the input data into a number of predefined categories. The learning algorithm is presented with a large volume of labelled text data, and the algorithm then learns a function which maps the data to its labels based on the contents of the input text. Once this function is learnt the algorithm should be able to correctly classify a new unseen instance of the data. Document clustering is an instance of unsupervised learning where the text data is presented without any labelling. The algorithm is then tasked with grouping this data such that similar data are in the same group, while dissimilar data are in distinct groups.

Numerous algorithms [1] have been proposed for general purpose classification and clustering. Many of these have been applied on text documents and have achieved varying degrees of success. The data points are assumed to exist in an 'input space' and either a separation (classification) or a grouping (clustering) which satisfies the 'hypothesis' is searched for in this space. Thus, the representation of the input space plays a major role in the efficacy of the machine learning algorithms.

Both classification and clustering algorithms require the input data be transformed to a representation which highlights the characteristics of the underlying data. This is done because dealing with raw untransformed data is cumbersome as it would necessitate searching through a complex hypothesis space. The transformation reduces the volume of data to be processed resulting in better speed and accuracy.

The vector space model [2] is the most commonly used framework to model this trans-

formed data. In this model, each characteristic or attribute of the input data is considered to be an axis of a multidimensional vector space. The data is represented as vectors whose components are made up of these attributes. These vectors are assumed to exist in this space and therefore reveal the proximity structure of the data. Thus the choice of attributes is critical in the success of the vector space model.

Text data is composed of words grouped into sentences and paragraphs. One technique to represent text data using the vector space model breaks down the textual data into a set of words. Each of these words corresponds to an attribute of the input data and therefore becomes an axis in the vector space. The data is then represented as vectors, whose components correspond to the words contained in the data collection and whose value indicates either a binary present/absent value or a weightage for the word for the data point. This representation is known as the 'Bag of Words' (BoW) representation and is the most commonly used representation for text data.

The English language typically contains 200,000 to 300,000 distinct words which in theory could be used as components in the vector space. The number of components used to represent the document is known as dimensionality. In practise, the dimensionality is of the order of a few tens of thousands. The space formed by such high dimensions tends to be huge. The documents are scattered in this large volume space resulting in a sparse representation. This seriously affects the performance of the machine learning algorithms and is known as the curse of dimensionality and is one of the deficiencies of the BoW scheme. The curse of dimensionality creates a serious bottleneck for most machine learning algorithms.

Using just the raw words as features ignores a lot of information contained across sentences and paragraphs in textual data. Moreover, almost all English words have many senses, *i.e.*, the same word can be used in different contexts to mean different things. The BoW scheme tends to ignore these characteristics of textual data.

A semantically motivated representation would be better at capturing the underlying characteristics of the data than frequency based methods such as BoW which is the current norm. This thesis introduces a novel technique to represent documents using lexical chains, which results in a semantically motivated representation.

We hypothesize that using an alternative representation like lexical chains, which addresses some of the problems of the BoW scheme, will be advantageous for machine learning tasks on text documents. Ideally such a representation should have the following characteristics:

- Use more semantics in the representation
- Reduce the dimensionality of the feature vectors formed
- Be amenable to the vector space model so that existing machine learning algorithms can use them

In this thesis, we propose the use of lexical chains as document features. The use of lexical chains reduces the dimensionality of the feature space significantly, while introducing much more semantics in the representation than the traditional bag of words. This, as we show in subsequent chapters, results in better performance as well as achieves significant improvement in the runtime performance.

Lexical chains are groups of words which exhibit a 'cohesion' among them. It is based on the idea that semantically related words co-occur more than 'just by chance'. By identifying and using lexical chains, it would be possible to represent the document in terms of their concepts. Moreover, since the concept space will be much smaller than the term space, the dimensionality will also be reduced.

1.1 Objectives and Scope

The objective of this thesis is to investigate and develop a semantically motivated representation for documents which will be useful for machine learning tasks. We are interested in developing a representation which can be widely used. The objectives are summarised below:

- Investigate and develop a semantically motivated representation scheme for documents
  - The representation should be amenable for use in general machine learning algorithms
  - The representation should, if possible, result in a lower number of dimensions than existing representations
  - The representation should perform better than existing representations
- Investigate and develop a soft clustering algorithm for documents, where documents can be present in multiple clusters

Most machine learning algorithms assume that the input data is represented in the vector space model. Therefore, it is essential that our representation follows this model, so that existing machine learning algorithm can be applied on it straightaway. In the next section, we highlight one of the key problems with the current representation. Documents contain multiple strands of information. Most of the popular clustering algorithms are 'hard' clustering algorithms where each document is placed in a single cluster. This dilutes the inherent nature of the information contained in textual data. Therefore it would be ideal if documents can be clustered in a 'soft' manner in which a document can be placed in multiple clusters. This, we feel, will better represent the underlying distribution of topics in a document.

The scope of this thesis is limited to developing a semantically motivated representation and proving its advantage over traditional representation schemes for machine learning tasks.

Most of the work done in this thesis is on clustering, while we also show that it works for classification as well.

The plan of this thesis is as follows: The rest of this chapter introduces the classical 'Bag of Words' features for documents and describes the standard procedure used to compute the BoW features. Chapter 2 introduces the notion of lexical chains in detail. We propose some significant deviations from the traditional way of computing lexical chains culminating in a new and improved algorithm to compute lexical chains. Chapter 3 shows how lexical chains can be used to represent document features and proposes five schemes for weighting the lexical chain features. Chapter 4 evaluates lexical chain based features on two machine learning tasks - clustering and classification. Chapter 5 introduces an approach for soft clustering documents using lexical chains and Chapter 6 summarises the conclusions.

1.2 Bag of Words Representation

The vector space model is, in general, the universal model representing the data in almost all machine learning algorithms. The model works on the premise that 'spatial proximity implies semantic proximity' [3]. Representing data in its original domain is cumbersome. Instead, measurements are made on the data and are used to characterise it. These measurements are called 'features'. Each data point is then represented as a vector of attributes or features. Each component of the vector corresponds to a feature and its value corresponds to the value of the measurement. These feature vectors exist in a high dimensional space where similar points will appear close together while dissimilar points will be distant from each other.

The Bag of Words representation is a simple and popular scheme for representing documents in a vector space model. In this scheme, each feature corresponds to a term present in the document collection. The measurement corresponding to each feature is usually either a binary 'present/absent' value or the frequency of the term in the document.

In theory, every distinct term present in the collection could be used as a feature. But this results in a vector space with an unnecessarily large number of dimensions. Thus, in practise the following steps are followed for computing the Bag of Words representation.

Tokenisation This step involves the use of a standard lexical analyser to tokenise the input documents and extract the numbers, words and punctuations present in the document. Usually for machine learning tasks, only the words are used as features and numbers and punctuations are discarded.

Stop Word Removal This step involves the removal of words such as '*this, the, that, where, what, when, is, and, etc.,*' which appear very frequently and carry no (or very little) semantic information. In addition, it is also common to remove words which occur very frequently (*i.e.*, in nearly all documents) in a collection, as these words have very little or no discriminative power.

Stemming This step helps to reduce the number of distinct terms by conflating words such as *goes, going, gone* into a single word *go*. Various algorithms for stemming have been proposed for the purpose. The work described in this thesis uses a Porter Stemmer [4] for stemming the terms, since it is the most widely used algorithm reported in the literature.

Term Weighting This step computes the 'importance' of each feature present in a document. Numerous variations have been proposed and used to varying degrees of success. The simplest term weighting scheme is the binary scheme in which the component corresponding to a feature is set to 1 if the term is present in the document and to 0 otherwise. Another variant is called the term frequency (tf) where the component corresponding to a feature gives the frequency with which it occurs in the document. A better variant is called the 'tf.idf' scheme where the tf is

weighted by the idf or the inverse document frequency [3] which is the number of documents in which the term occurs.

We illustrate the above steps using an example. Consider the following two sentences:

"*Give a man a fish; you have fed him for today.*
Teach a man to fish; and you have fed him for a lifetime"

After tokenising and removing stop words from the above sentences, we get *give, man, fish, fed, today, teach, lifetime* as possible features. These words are already in their root form and therefore stemming them will result in the same set of features. Thus, these features will form the components of the feature vector. Treating each sentence as an independent data point, we get two feature vectors, each containing 7 components - [give, man, fish, fed, today, teach, lifetime]. Using the binary weighting scheme, these feature vectors are [1,1,1,1,1,0,0] and [0,1,1,1,0,1,1].

One evident problem with the BoW scheme is that it treats the words 'fish' in both the sentences as a single dimension. But the sense in which the word is used in the two sentences is considerably different. While in the first sentence it refers to an animal, in the second it refers to an occupation or an activity. This results in smudging the topics, leading to imprecise results. The above is a very difficult example where the context gives very little information about the variation in the meanings of the words, but on an average the task is more tractable. Thus there exists a need for a more semantic representation for textual data, which will capture and represent the semantics of the underlying content.

CHAPTER 2

Lexical Chains

Language, both written and spoken, is a means by which humans communicate their thoughts and ideas. Language is composed of sentences, which in turn are composed of words. Words carry meaning, and a combination of words put together in a certain manner forms a concept. For example, the words *apple* and *red* refer to a *fruit* and a *colour* respectively. When we combine these two words together to form the sentence - '*Apples are red*', we convey the concept of apples being red in colour. But the sentence does not explicitly state that the 'colour' of the apple is red. But we effortlessly infer that it is the colour and not the size or the shape that is being referred to. How do we do it?

One explanation is that there exists a knowledge base in our minds, which we acquire over time, which gives us this information. This knowledge base helps us recall that apples have colour and that red refers to a colour and therefore the above sentence refers to the colour of the apple. Thus, the context in which the words are used will determine the semantics of the sentence.

Not all combination of words give rise to meaningful sentences, *i.e.*, a random collection of words cannot form a meaningful text. This fact was famously illustrated by Noam Chomsky with the sentence "Colourless green ideas sleep furiously". This sentence is grammatically correct but is meaningless. Thus, for a sentence to be meaningful, the words used in them should be correlated to the concept being described, *i.e.*, the words used should describe some aspect of the concept being conveyed. The same applies to higher level constructs such as paragraphs, chapters, and documents as well. A paragraph of any meaningful text is not just a set of random sentences, but a set of correlated sentences.

Halliday and Hasan [5] formalised this notion of correlation between words in a meaningful unit of text with the concept of 'lexical cohesion'. Lexical cohesion is the cohesion which arises from the semantic relationships between words. This property results in semantically related words occurring together in a passage. Lexical cohesion can arise due to various characteristics of the words in a sentence. Halliday and Hasan [5] classified lexical cohesion into five classes as follows:

1. Reiteration with identity of reference
2. Reiteration without identity of reference
3. Reiteration by means of super ordinate
4. Systematic semantic relation
5. Non-systematic semantic relation

The first three relations involve reiteration which includes repetition of the same word in the same sense (*e.g.*, car and car), the use of a synonym for a word (*e.g.*, car and automobile) and the use of hypernyms (or hyponyms) for a word (*e.g.*, car and vehicle) respectively. The last two relations involve collocations *i.e.*, semantic relationships between words that often co-occur (*e.g.*, football and foul). A systematic relationship is a categorical relationship. An example of a systematic relationship are words such as *one, two, three* or *red, green, blue*. A non-systematic relationship is one in which words co-occur due to their lexical context. Words such as *car, tyre, engine* are non-systematic relationships.

The concept of *lexical chains* was first proposed by Morris and Hirst [6]. It was based on the premise that lexically cohesive words co-occur together and span across sentences and paragraphs. They hypothesised that if such cohesive words can be grouped together, they form topical units. Each such topical unit is called a lexical chain. Lexical chains are an indicator of the discourse structure of text, *i.e.*, they reflect the pattern of topics and subtopics in a document.

Their work on lexical chains was inspired by Halliday and Hasan's work [5] on lexical cohesion and the theory of discourse structure proposed by Grosz and Sidner [7]. They gave an algorithm to compute lexical chains using the Roget's Thesaurus and also showed its correspondence to the discourse structure of text. Subsequently numerous researchers (see Section 2.2) have used lexical chains for various applications such as text summarisation, malapropism detection and hyper link generation.

Definition 1 *A lexical chain is a set of semantically related words which occur in a unit of text.*

Two words are semantically related to each other if they are:

- Identical and used in the same sense, or
- Synonymous, or
- One is an immediate generalisation (or specialisation) of the other, or
- Share a common parent in a generalisation hierarchy.

Naturally, computing lexical chains requires the availability of a database which identifies and records the semantic relations between words. Such databases are known as 'thesaurus'. Thesaurus are designed for human use and are not very suitable for use by a computer. Moreover, the variety of relations between words recorded in a thesaurus are limited. More recently, a computationally tractable database known as WordNet [8] has been gaining popularity and is now the main database used to compute lexical chains.

2.1 An Example

We illustrate the concept of lexical chains using an example. Consider the sample of text taken from a document in the 20 Newsgroups (20NG) [9] dataset, shown in Fig. 2.1. A few of the

Has anybody noticed that Toyota has an uncanny knack for designing horrible ugly station wagons? Tercels, Corollas, Camrys. Have their designers no sense at all?

Figure 2.1: Contents of file 101574 in rec.auto group from 20 Newsgroups Corpus

bent, hang, knack
design, designing
architect, designer
sense, signified

Table 2.1: Some of the Lexical Chains formed from the text in Fig. 2.1

lexical chains which occur in this document are shown in Table 2.1.

Each lexical chain consists of one or more words which occur in or across documents and are semantically related. For example, the words *bent, knack, hang* refer to the concept of "a special way of doing something". Each of the chains in Table 2.1 give a good indication of a topical strand contained in the document. In effect, each lexical chain maps onto a topic or a concept. It is possible to infer from just the four chains, that the document is speaking about *skill, designs, designers*, and *sense*. This is a more direct mapping to the concept space than the bag of words, where one would use each of the terms present in the document, and then try and infer the concept of the document.

2.2 Previous Work

Lexical chains have been reported in the literature as an intermediate representation for various applications such as automatic text summarisation [10, 11], malapropism detection and correction [12] and hypertext construction [13]. More recently Ercan, *et. al.* [14], reported using lexical chains for keyword extraction. In a similar application, Stokes [15] uses lexical chains for new event detection.

The concept of lexical chains was introduced by Morris and Hirst [12] and was based on

the Roget's Thesaurus. Since at that time the Roget's Thesaurus was not available in electronic form, a computer realization of their algorithm was not possible. The release of the WordNet database was a milestone in this field and resulted in many new lexical chain algorithms using WordNet. We describe here the major lexical chaining algorithms reported in the literature.

Hirst and St-Onge [12] qualify relations into extra-strong (identity and synonymy), strong (hypernymy and hyponymy) and medium strong (hypernymy, hyponymy path). Given a word, they try to identify a relation with a chain in the following order of importance - *extra strong*, *strong* and then *medium strong*. They employ a greedy strategy to disambiguate between the chains *i.e.*, words are added to the chain with which they have the strongest relation.

Barzilay and Elhadad [10] showed that chain selection cannot be done through a greedy approach. They proposed using the "whole picture" of the chain distribution in order to select the right chain. In their approach they create a component which is a list of interpretations which are exclusive of each other. As each new word is encountered, it is assessed to determine if there exists a relation with any of the existing components. Failing this, a new component is created. If a relation exists, all possible interpretations in the matching component are created. They then compute the score of an interpretation using an empirically derived scoring scheme based on the number and weight of the relations between the chain members. They define the best interpretation as that interpretation with the most connections. Since in this approach the number of interpretations can grow rapidly, they maintain a threshold beyond which they prune out the weak interpretations.

Silber and McCoy [11] propose a two pass algorithm for computing lexical chains. They propose four relations - identity, synonymy, hypernym/hyponym relation and sibling hypernym/hyponym tree relation. They employ a part-of-speech tagger to identify the noun instances within the document. In the first step, for each noun instance encountered, each sense of the noun instance is placed into every "meta-chain" for which it has an identity, synonym or hypernym relation with that sense. These meta-chains represent every possible interpretation of

the text. In the second step, they find the best interpretation by taking each noun and looking at all its meta-chains. A score is computed based on the type of relation and distance factors to identify which meta-chain the nouns contribute the most to and then delete the noun from the other meta-chains.

Jarmasz and Szpakowicz [16] use a computerised version of the 1987 edition (released almost 10 years later) of Penguin's Roget's Thesaurus of English Words and Phrases [17] called the Electronic Lexical Knowledge Base (ELKB) [18] to build lexical chains. Their algorithm builds *proto-chains*, a set of words linked via thesaurus relations. They then score the proto-chains using a scheme similar to that of Silber and McCoy [11] to obtain the final lexical chains.

Lexical chains, to the best of our knowledge, have never been used to represent documents for machine learning tasks. Moreover, our algorithm features some improvements over the previous algorithms. Most of the previous algorithms try to disambiguate the sense of the word during the chain formation process. This complicates the lexical chain computation. One of the key differences in our algorithm is that we separate out the disambiguation process as a separate pre-processing step using available stand alone disambiguation algorithms. This significantly simplifies the algorithm. We describe this step in detail in Section 2.5.

Another significant aspect on which our algorithm differs is spatial proximity. The algorithms reported in the previous literature consider the spatial distance between words when forming the chains. That is, a word is added to a chain only if it occurs within a certain number of sentences from the previous occurrence of the chain. This was feasible because the previous algorithms operate only within a document. Our approach relies on the lexical chains being computed across documents. This precludes the use of spatial proximity because it is difficult to define spatial proximity across documents. But as we will see later, this relaxation does not seem to affect our results at all.

2.3 Lexical Chains and Discourse Structure

Discourse structure refers to the sequence of topics and subtopics contained in a portion of text (paragraph, document, *etc.*). A computational theory of discourse structure for general text was proposed by Grosz and Sidner [7]. According to them, discourse structure is composed of three interacting components - linguistic structure, intentional structure and attentional state.

Linguistic structure divides the discourse into (possibly overlapping) segments which consist of groups of sentences. Each of these segments will refer to a topic. Linguistic structure forms the basic unit of the discourse structure. Intentional structure refers to the purpose of the discourse. Each of the linguistic structure units has a purpose which specifies what the intention of the unit is, in the discourse. These in turn add up to form the purpose of the overall discourse. Attentional state refers to the things or objects that the discourse is focussed on in each segment.

As noted by Grosz and Sidner, the identification of linguistic structure is the key to the theory of discourse structure. Morris and Hirst showed that lexical chains are capable of giving reasonably good indicators of linguistic structure in a document. More precisely, they showed that when a chain ends at a segment it implies the end of the topical strand and if a new chain is started it indicates a new topical strand. If an old chain is resumed (*i.e.*, referred to again in a passage), it implies the resumption of an earlier topical strand.

Our work exploits this characteristic of lexical chains - each lexical chain is an indicator of a topical strand and the segments connected by it belong to the same topic. We work on the premise that, instead of just computing the lexical chains with respect to a single document, if we compute the chains across documents, we are in effect chaining together those documents belonging to a topical strand. Therefore, the set of lexical chains contained in a document could give an indication of the set of topics contained in it. Thus, representing documents in terms of lexical chains would be equivalent to representing it using the underlying topics. This, we

hypothesise will be a much better representation for documents than using a bag of words. This hypothesis has been ratified by our experimental results discussed later in the thesis.

Before we describe our algorithm to compute lexical chains, we diverge to describe WordNet which is the core of our algorithm and word sense disambiguation which is a critical component of our algorithm.

2.4 WordNet

WordNet [8] is a linguistic database created at the Cognitive Science Laboratory [19] of Princeton University with the goal of creating a machine tractable model of human language processing capability. WordNet organises words into groups known as *synsets*. Each synset contains a group of synonymous words and collocations[1] and corresponds to a concept. Thus, WordNet can be considered to be a database which maps words to their semantic concepts. In addition, each synset also contains pointers to other semantically related synsets. WordNet has been in development for more than 20 years and the latest version (v3.0) of the database contains 155,287 words organised in 117,659 synsets for a total of 206,941 word-sense pairs.

WordNet has four categories of words - nouns, verbs, adjectives and adverbs. Within each category, the words are organised into synsets. Each synset is identified by a unique synset number. Each word belongs to one or more synsets with each instance corresponding to different senses of the word and are numbered according to their frequency of occurrence in real world usage. The most frequent sense of a word is given a sense number of 1 and for each subsequent sense the sense number is incremented by one. Thus, any word in WordNet can be uniquely referred to using a 3-tuple consisting of the word, its part of speech and its sense number. Each such tuple corresponds to a synset in WordNet. An example of a synset in WordNet is shown in Fig. 2.2.

[1] a collocation is a sequence of words that go together to form a specific meaning, such as "car pool"

bent, knack, hang – (a special way of doing something; “he had a bent for it”; “he had a special knack for getting into trouble”; “he couldn’t get the hang of it”)

Figure 2.2: An example of a WordNet Synset

In addition to synonymous words, each synset also provides:

- a definition for the concept
- an example called a ‘gloss’ showing how the words in the synset are used in a particular context
- one or more links to other synsets which are related to it

WordNet defines a number of semantic relations which are used to connect a synset to other related sysnets. This forms a semantic net relating a word to other related words. These relations vary based on the type of word, and include:

hypernyms A word X is a hypernym of word Y, if X is a generalisation of Y, *i.e.*, Y is a kind of X. For example, the word *vehicle* is a hypernym of *car*, since a *car* is in general a *vehicle*.

hyponyms A word Y is a hyponym of word X if Y is a specialisation of X. For example, the word *hatchback* is a hyponym of the word *car*, since a hatchback is a specific type of car.

coordinate terms Words X and Y are coordinate terms if X and Y share a hypernym, *e.g.*, car and motorcycle are both types of motor vehicles.

holonym Word X is a holonym of word Y, if Y is a part of X, *e.g.*, car is a holonym of tyre.

meronym Word X is a meronym of Y, if X is a part of Y, *e.g.*, tyre is a meronym of car.

troponym The verb Y is a troponym of the verb X, if the activity X and Y are the same, *e.g.*, throw and hurl are troponyms

antonym Word X is the antonym of word Y, if Y is the opposite of X, *e.g.*, go and come are antonyms

The synsets together with the semantic relations connecting them can be considered to be a hybrid of a dictionary and thesaurus. This forms a powerful knowledge base of information about the words and can be exploited for various natural language processing applications. WordNet has been used to perform Word Sense Disambiguation (WSD) described in the following section. The lexical chain computation algorithm described later in this chapter also uses WordNet to identify the relations between lexical chains and words.

WordNet also includes a morphology database which can be used to infer the root form of a word from its inflected form. Though WordNet contains a large number of common words, it does not cover domain specific vocabulary. WordNet is also available in numerous other languages [20]. While the work done in this thesis limits itself to English, we feel that the algorithm and approach used in this thesis can be easily extended to other languages.

2.5 Word Sense Disambiguation

Almost all words in the English language are polysemous, *i.e.*, they have multiple meanings or senses. But any instance of the word will refer to only one of those multiple meanings. The precise meaning of a word is dependant on the context in which it is used, *i.e.*, the words surrounding it in a sentence. For example, the word *bank* could mean a river bank, a financial bank or the banking on the road. Usually, such polysemous words are used without any qualifiers which specifies the sense in which the word is used. But, humans can effortlessly disambiguate the meaning of the word just by looking at the context in which they appear.

The same is not true of computers. An active domain of research on doing such disambiguation exists and is formally known as Word Sense Disambiguation (WSD). The goal of

WSD is to automatically disambiguate the meaning of the word from its context, *i.e.*, identifying the sense in which a word is used and assigning that sense to it. WSD finds applications in many areas of natural language processing such as machine translation, question-answering, information retrieval, *etc*.

Numerous algorithms have been proposed for WSD, and recently the performance of these algorithms have improved considerably. An extensive survey of the work done in this field can be found in [21]. WSD algorithms can be divided into three broad categories:

Dictionary based methods These methods use a lexical knowledge base (*e.g.*, dictionary, WordNet, *etc.*) to perform WSD. The knowledge base provides a sense inventory and the words are mapped on to one of its senses contained in the dictionary.

Supervised methods These methods make use of sense-annotated corpora to learn a mapping from context to word senses.

Unsupervised methods These methods work directly from raw unannotated corpora by creating context groups of words occurring in various contexts.

We use a dictionary based method for the work done in this thesis. Dictionary based WSD methods disambiguates a word with reference to a dictionary. A dictionary collates the various senses of a word. The dictionary used can be either a conventional dictionary, a thesaurus or WordNet which is the more popular option. These methods find a mapping between a word and one of the various senses found in the dictionary on the basis of some heuristics.

WordNet categorises words into one of four possible parts of speech - noun, verb, adjective and adverb. Within each part of speech, it is further divided into its possible senses. Each sense corresponds to a meaning. For example, in WordNet, the word ‘bank’ has eighteen senses across two parts of speech namely nouns and verbs as shown in Fig. 2.3. The serial number against each entry is the sense number of the word in that particular part of speech (pos).

Many dictionary based WSD algorithms have been proposed using WordNet. A very popular algorithm is the Lesk Algorithm [22]. This algorithm disambiguates a word by comparing the WordNet gloss of the word with those of the surrounding words within a window. The word is assigned the sense corresponding to the gloss which has maximum overlap with the other words in the window. This algorithm is based on the heuristic that the meaning of a word will be closely related to the meaning of the neighbouring words, and that this can be inferred from the amount of overlap in the definitions of these words.

For example, consider the following sentences:

- I went to the **bank** to cash a check
- The boat anchored on the **bank** of the river

As shown in Fig. 2.3, the noun form of the word 'bank' has 10 senses. The gloss for sense number 1 contains the words 'check' and 'cashed' and overlaps with the other words in the first sentence. Such an overlap is not observed with any of the other senses. Thus, the heuristic concludes that the word 'bank' in the first sentence is used in a manner similar to sense number 1. Similarly, the words in the second sentence overlaps with the gloss of sense number 2 of the word 'bank' and we can conclude that the second instance of the word 'bank' corresponds to sense number 2.

Banerjee and Pederson [23] observed that the glosses corresponding to each sense tend to be short, with an average length of 7 in WordNet. The disambiguation decision in the previous example had to be made on the basis of a very small overlap. This constrains the disambiguation algorithm. They proposed an improved algorithm which they called as the Extended Lesk Algorithm.

The Lesk algorithm works on the premise that when two words which appear close to each other in a sentence and their glosses share common words, then they are being used in a similar

Relation A	Relation B
hypernym	holonym
hypernym	hypernym
hyponym	hyponym

Table 2.2: Example of extended Lesk relations

sense. Higher the number of common words in a pair of glosses, the higher is the semantic relatedness of the two words. Banerjee and Pederson [23] extended this measure in two ways. Firstly, instead of using just the two glosses from the two words being compared, they included glosses from other words related to the target word. These other synsets considered, share a WordNet relation with the target synsets. A small sample of the relations that can be used is shown in Table 2.2. So in addition to computing the overlap between the glosses of two target words, the extension computes the overlap between the gloss of the hypernym of the first target word and the gloss of the holonym of the second target word and so on. This expands the size of the glosses used to compute the overlap and provides a solution to the problem of short glosses. This they felt would give a more precise disambiguation. Secondly, they also suggested that instead of using just the number of common words as the measure, it would be better to use the number of common phrases. A common *n* word phrase would be a more rare occurrence than single words and hence they hypothesised that it would be a better indicator of similarity and gave it a higher weightage.

The working of the algorithm is as follows: Given a pair of words, all the synsets containing each of the words are obtained. For each pair of synsets, the algorithm computes the overlap of the glosses of synsets related to it. The 'relatedness' refers to the WordNet relation. For example, consider the relations listed in Table 2.2. Given two words x and y, we obtain from WordNet all synsets containing x and y and denote these sets as $X = \{X_i \mid X_i \; is \; a \; synset \; in \; WordNet \; and \; x \; \epsilon \; X_i\}$ and $Y = \{Y_j \mid Y_j \; is \; a \; synset \; in \; WordNet \; and \; y \; \epsilon \; Y_j\}$. Each of the synsets in X and Y correspond to a different sense of x and y respectively. The extended Lesk algorithm will take a pair of

synsets $(X_i,\ Y_j)$ and compute the overlap between the glosses of the synsets corresponding to a pre-defined set of pairs of relations similar to that shown in Table 2.2. For example, using the pairs shown in Table 2.2 the overlap between X_i and Y_j is given by

$$\begin{aligned} overlap(X_i, Y_j) = overlap\ (hype(X_i),\ holo(Y_j)) + \\ overlap\ (hype(X_i),\ hype(Y_j)) + \\ overlap\ (hypo(X_i),\ hypo(Y_j)) \end{aligned}$$

The function $hype(X)$ corresponds to a WordNet relation and returns the concatenated glosses of all synsets related to X by the specified relation, which in this case is hypernymy. The words x and y are assigned the senses corresponding to i and j for which $overlap(X_i, Y_j)$ is maximum among all i's and j's. The output of this algorithm assigns to each words from the input, a part of speech from one of the four parts of speech in WordNet, *i.e.*, nouns, verbs, adjectives and adverbs and a sense number corresponding to the synset sense numbers in WordNet.

This technique for disambiguating two words can be extended to a whole document by sequentially running it on a moving window of an odd number n of words including the target word, with $(n-1)/2$ words before and after the target word. A higher value of n results in better accuracy but lower speed and vice versa. In our experiments, n was set to 5.

In this chapter, we have described the concept of lexical chains, and detailed two critical components which will be used to compute lexical chains. In the next chapter, we show how these can be used to compute lexical chains.

```
The noun bank has 10 senses (first 9 from tagged texts)
1. depository financial institution, bank, banking concern,
   banking company -- (a financial institution that accepts
   deposits and channels the money into lending activities;
   "he cashed a check at the bank"; "that bank holds the
   mortgage on my home")
2. bank -- (sloping land (especially the slope beside a body
   of water); "they pulled the canoe up on the bank"; "he sat
   on the bank of the river and watched the currents")
3. bank -- (a supply or stock held in reserve for future use
   (especially in emergencies))
4. bank, bank building -- (a building in which the business
   of banking transacted; "the bank is on the corner of Nassau
   and Witherspoon")
5. bank -- (an arrangement of similar objects in a row or in
   tiers; "he operated a bank of switches")
6. savings bank, coin bank, money box, bank -- (a container
   (usually with a slot in the top) for keeping money at home;
   "the coin bank was empty")
7. bank -- (a long ridge or pile; "a huge bank of earth")
8. bank -- (the funds held by a gambling house or the dealer
   in some gambling games; "he tried to break the bank at
   Monte Carlo")
9. bank, cant, camber -- (a slope in the turn of a road or
   track; the outside is higher than the inside in order to
   reduce the effects of centrifugal force)
10. bank -- (a flight maneuver; aircraft tips laterally about
    its longitudinal axis (especially in turning); "the plane
    went into a steep bank")

The verb bank has 8 senses (first 2 from tagged texts)
1. bank -- (tip laterally; "the pilot had to bank the aircraft")
2. bank -- (enclose with a bank; "bank roads")
3. bank -- (do business with a bank or keep an account at a bank;
   "Where do you bank in this town?")
4. bank -- (act as the banker in a game or in gambling)
5. bank -- (be in the banking business)
6. deposit, bank -- (put into a bank account; "She deposits her
   paycheck every month")
7. bank -- (cover with ashes so to control the rate of burning;
   "bank a fire")
8. trust, swear, rely, bank -- (have confidence or faith in; "We
   can trust in God"; "Rely on your friends"; "bank on your good
   education"; "I swear by my grandmother's recipes")
```

Figure 2.3: Listing for the word 'bank' from the WordNet database

CHAPTER 3

Lexical Chains as Document Features

In this chapter, we describe our algorithm for computing lexical chains and describe how lexical chains can be used as document features. We also describe two different chain selection schemes.

3.1 Computing Lexical Chains

Morris and Hirst [6] suggested that in addition to being able to model discourse structure, lexical chains can also disambiguate words, *i.e.*, the chain forming process includes an implicit WSD process. Accordingly the various algorithms in literature propose various heuristics to perform WSD during the chain forming process, while lamenting about the difficulty of doing so.

We feel that with the advance in research in the field of WSD and the availability of good WSD algorithms, there is no longer a need for interleaving the WSD process with the lexical chain formation process. We can greatly simplify the algorithm to compute lexical chains by separating the WSD process from the chain computation process. One of the distinguishing features of our algorithm is that we use a full-fledged WSD algorithm to explicitly disambiguate the senses of the words in a document as a preprocessing step before computing the lexical chains. We use the Extended Lesk Algorithm described in Section 2.5 for the purpose. Our lexical chaining algorithm is structured in such a way that the Extended Lesk Algorithm can easily be replaced with any another WSD algorithm which maps words to WordNet senses.

Not all words in a document can be used to form lexical chains. At a bare minimum, stop words must be excluded as they will be meaningless. We refer to those set of words in a document which are used to compute lexical chains as candidate words. We first run the WSD algorithm on the input document as a preprocessing step. This will markup each word (other than stopwords) in a document using its part of speech and sense number from WordNet. The resulting words are filtered to obtain the set of candidate words which are used to form lexical chains. This filtering can be done in various ways to obtain different variants of lexical chains.

Silber and McCoy [11] suggested using only nouns and compound nouns present in a document, for computing lexical chains. This is because most of the concepts are referred to by nouns. The other researchers are silent on the issue but seem to prefer only nouns. We wanted to test this hypothesis and created 3 categories of candidate words - N (Nouns only), NV (Nouns+Verbs), and NVA (Nouns+Verbs+Adjectives+Adverbs). We later demonstrate empirically that Nouns alone are adequate to represent the information contained in the documents.

Lexical Chains can be computed at various granularities - across sentences, paragraphs or documents. In general, to compute lexical chains, each candidate word in a unit of text is compared with each of the lexical chains identified so far. If a candidate word has a 'cohesive relation' with the words in the chain it is added to the chain. On the other hand, if a candidate word is not related to any of the chains, a new chain is created and the candidate word is added to it.

Each lexical chain is identified by a unique numeric identifier and contains a set of semantically related words. Each word in a chain is represented as a 4-tuple <term, pos, sense, rel>, where 'term' is the actual word, 'pos' is the part-of-speech of the term, 'sense' is the WordNet sense number and 'rel' is the relation of this word to the chain. In theory any of the WordNet relations can be used to relate candidate words to the chains.

Hirst and St-Onge [12] suggested giving weights to the various relations with identity and

synonymy relations having the highest weights, immediate hypernymy and hyponymy having intermediate weights and sibling hypernymy or hyponymy having the lowest weight. As per this scheme, a word is added to the chain with which it is related with the highest weight. According to Morris and Hirst [6], using relations other than identity and synonymy will imply a transitive relation between words and chains, *i.e.*, A is related to B and B is related to C, and therefore A is related to C. Thus including hypernyms and hyponyms relations in a chain will result in the chains representing composite topics. This also reduces the granularity of the topic contained in each chain. For example, when hypernyms are used to compute lexical chains, the words 'motor vehicle', 'car' and 'bike' will result in a single chain since 'motor vehicle' is a hypernym of both 'car' and 'bike' when in reality they are distinct but related concepts.

Thus, we feel that using only the identity and synonymy relations will result in crisp chains, each of which represents a single topic. We feel that for generating feature vectors it will be better to use only the identity and synonymy relations to form the chains. This hypothesis is tested and validated in the next chapter where we compare the performance of lexical chains computed using other WordNet relations.

3.1.1 Global Set

One of the distinguishing features of our algorithm is that we compute lexical chains across documents. This requires us to store the identified chains, as well as provide a mechanism to create new chains. We facilitate this by introducing a data structure which we refer to as *Global Set*. The global set is a set of lexical chains which appear in at least one document. The global set maintains the following information:

- a list of documents processed
- the chains which appear in at least one document

- the number of times a chain occurs in a document

This information is useful when computing feature vectors. We have also implemented a serialisation mechanism on the global set data structure. This allows us to save and reuse a global set allowing us to run the lexical chaining algorithm in an incremental manner on many sets of documents.

3.1.2 Algorithm

Our algorithm to compute lexical chains is shown as Algorithm 1. The algorithm maintains a Global Set. The global set can either be initialised to a NULL set or can be bootstrapped with a previously computed global set. The documents are pre-processed using the Extended Lesk Algorithm. The disambiguated words are then filtered to obtain the candidate words for the documents.

Algorithm 1 Computing Lexical Chains

```
1:  Maintain a global set of lexical chains
2:  for each document do
3:      for each candidate word in document do
4:          Identify lexical chains in the global set with which the word is related
5:          if No chains are identified then
6:              Create a new lexical chain for this word and insert it into the global set
7:          end if
8:          Add the candidate word to the identified/created lexical chain
9:      end for
10: end for
```

Each candidate word is compared with each of the chains in the global set. If the word is related to a chain, then it is added to the chain. If no such chains are identified, then a new chain is created, the word is added to the chain and the new chain is inserted into the global set. This algorithm exhaustively computes all possible lexical chains for each document. We feel that it will be better to do chain selection on this global set of all possible chains, than to do it locally at each document. We use only the identity and synonymy relations to compute the

chains. A word has a identity or synonymy relation with another word, only if both the words occur in the same synset in WordNet.

3.2 Document Features

A feature vector is an n-dimensional vector of numeric values which is used to characterise a document. Each dimension in the feature vector corresponds to a feature which in our case is a lexical chain. The numeric values are called feature weights or simply weights.

The global set, after processing all the documents, contains the set of lexical chains which occur in one or more documents along with the other statistics mentioned in Section 3.1.1. This is pictorially shown in Fig. 3.1. Each lexical chain maps onto one or more documents and vice versa. From this mapping, we can obtain a list of documents contained in the global set. The set of all lexical chains which are used to represent the documents constitute the feature vector. The lower portion of Fig. 3.1 shows the feature vectors generated for the documents. The values $V_{1,1}$, $V_{2,1}$, $V_{m,3}$, *etc.*, corresponds to the feature weights as computed by one of the schemes described in the subsequent sections.

We introduce two categories of feature vectors - *Selected Chains* and *All Chains*. In the first category, we select and use a subset of 'good' chains from the set of chains assigned to a document, to represent it. This is inspired by the previous literature on lexical chaining algorithms, where chains are selected based on their 'strength'. This, as we will show later, results in small dimensional feature vectors and attains good performance as well. In the second category we use all the chains identified for a document to represent it. This is analogous to the bag of words approach. These schemes are elaborated below, while experiments to measure their utility are presented in the next section.

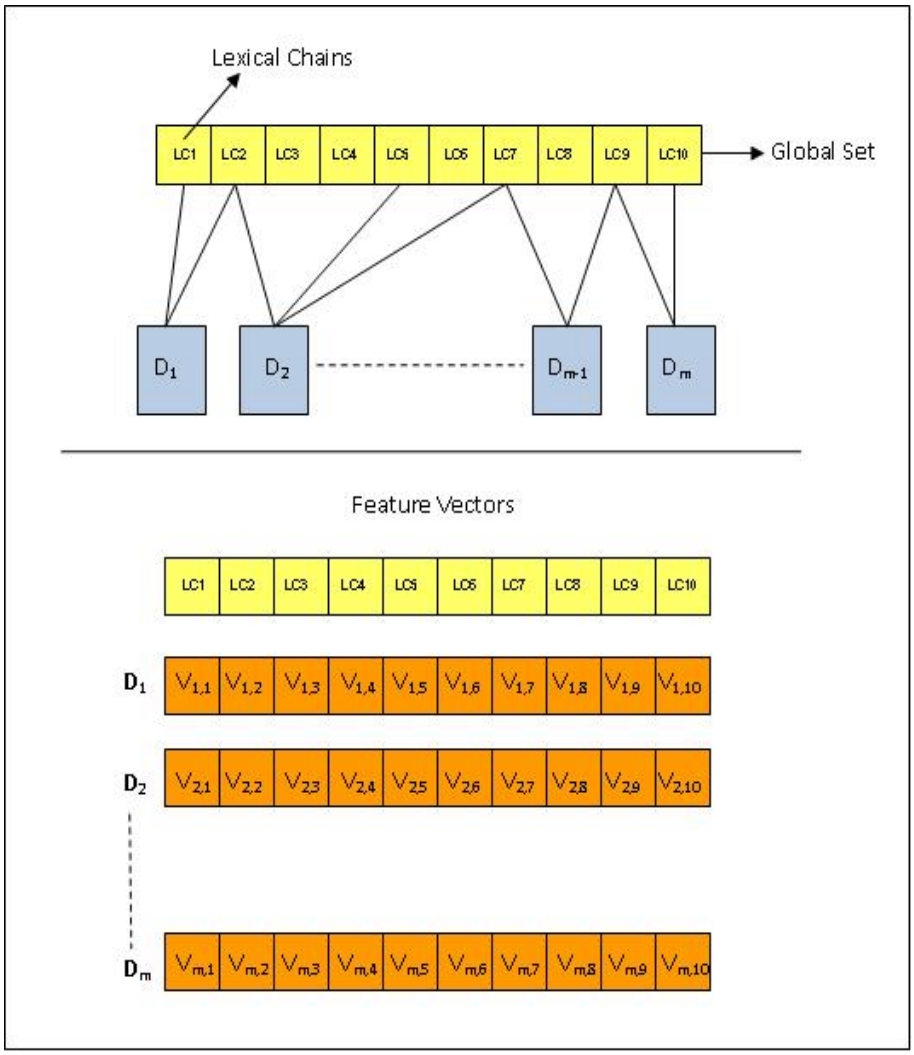

Figure 3.1: Schematic diagram of the Lexical Chain based Feature Vector Generation

3.3 Feature Vectors using Selected Chains

The algorithm described in the previous chapter, computes all possible lexical chains for a document. We hypothesise that it will be possible to identify and select a subset of these chains which are most representative of the contents of the document. This we feel would result in better discrimination between the documents and will reduce the dimensionality of the feature vectors.

In this scheme, we identify and use only a subset of 'good' chains to represent it. The

goodness of a chain is measured by the 'utility' of the chain to the document and is computed as follows:

Definition 2 *Length of a lexical chain L is defined as the number of words in the chain.*

$$length(L) = \text{Number of Words in Chain L} \tag{3.1}$$

The length of a lexical chain is an indicator of the strength of the chain in representing a topic and is the total number of words present in the chain including repetitions (words related by identity relationship). The length of a chain gives a composite measure of the number of documents in which the chain occurs and the number of occurrences of words in the chain in these documents.

The length of the chain can be used to determine topically significant chains from insignificant ones. Our intent is to generate features with good discrimination properties. For this we have to eliminate the extreme cases - extremely long chains and extremely short chains. The former case will occur when a chain appears in too many documents and the latter case occurs when the chain appears in too few documents. In both the cases, the discriminating power of the chain is considerably reduced and will add an unproductive dimension to our feature vector. We define a function $sig(L)$ to determine the significance of a chain based on its length. This is defined as:

Definition 3 *The significance of a lexical chain L in a global set G is defined as*

$$sig(L) = -\frac{length(L)}{\sum\limits_{l\ \epsilon\ G} length(l)} . \log_2 \frac{length(L)}{\sum\limits_{l\ \epsilon\ G} length(l)} \tag{3.2}$$

In effect, the value $sig(L)$ will peak for those chains which are not abnormally long or short with respect to the chains in the global set.

Given a chain L and a document we decide whether the chain should be used to represent the document based on the 'involvement' of the chain in the document. This is in effect the number of words in the document which are present in the chain. We define a binary function $related(W, L)$ which tells whether a candidate word W is present in chain L. This is defined as:

Definition 4 *A candidate word W is related to a lexical chain L, if W is present in* L.

$$\begin{aligned} related(W, L) &= 1,\ if\ W\ is\ present\ in\ L \\ &= 0,\ otherwise \end{aligned} \tag{3.3}$$

The utility of a chain L is a measure of how good a chain L will be in representing the document. This is based on the observation that this measure will prefer chains with high significance from the global set which are related to a large number of candidate words in the document. This function is defined as:

Definition 5 *The utility of a lexical chain L to a document D is defined as*

$$util(L, D) = sig(L). \sum_{all\ w\ \epsilon\ D} related(w, L) \tag{3.4}$$

We select and assign to the document, all those chains in the global set which cross a threshold on the utility of the chain. We obtained good results on setting the threshold as 'half the average' utility for a document. For a document D, let the set of all lexical chains assignable to D be $G' \subset \text{global set } G$. The threshold for D is computed as

$$threshold(D) = \frac{\sum_{L\ \epsilon\ G'} util(L, D)}{2\ .\ |G'|} \tag{3.5}$$

Ideally, we should have set a cut-off on the $sig(L)$ values and then used only those chains

which cross the cut-off to generate the chains. Instead, we avoid having to make a guess on the cut-off value and let the document 'decide' which chains from the global set should represent it using the $threshold(D)$ function. This results in an implicit 'cut-off' being applied on the $sig(L)$ values. The feature vectors formed will contain only those chains from the global set which have been selected by at least one document.

We use two feature weighting schemes to create the feature vectors:

Binary weighting

In this scheme, we put a 1 corresponding to a chain if the chain is assigned to the document and 0 otherwise.

$$\begin{aligned} value(L, D) &= 1,\ if\ lexical\ chain\ L\ is\ assigned\ to\ D \\ &= 0,\ otherwise \end{aligned} \tag{3.6}$$

We refer to feature vectors generated using this approach as *SelectedChains-Binary*.

Utility weighting

In this scheme, we use the value of $util(L, D)$ as the value for the component corresponding to each selected lexical chain. The utility function is a measure of how good the chain is for representing the document. We feel that this will be a good way for obtaining discriminating feature vectors.

$$\begin{aligned} value(L, D) &= util(L, D),\ if\ lexical\ chain\ L\ is\ present\ in\ D \\ &= 0,\ otherwise \end{aligned} \tag{3.7}$$

Correspondingly, we refer to the feature vectors generated using this scheme as *SelectedChains-Utility*.

3.4 Feature Vectors using All Chains

In this scheme, we use all the lexical chains in the global set to construct the feature vector. This follows from the usual approach in bag of words where all the identified words are used to create the feature vector. We use three different feature weights:

Binary weighting

This scheme is similar to that used in Section 3.3 and uses a 0/1 as the component value to signify the presence or absence of a feature. As before, the component corresponding to a lexical chain is 1 if the chain is present in the document and 0 otherwise leading to a fairly simple scheme for generating feature vectors.

$$\begin{aligned} value(L, D) &= 1,\ if\ lexical\ chain\ L\ is\ present\ in\ D \\ &= 0,\ otherwise \end{aligned} \tag{3.8}$$

We refer to this scheme as *AllChains-Binary*.

Utility weighting

This scheme is similar to that described in Section 3.3. As before, we use the value obtained from $util(L, D)$ as the feature weight if the chain is present in the document and 0 otherwise.

$$\begin{aligned} value(L, D) &= util(L, D),\ if\ lexical\ chain\ L\ is\ present\ in\ D \\ &= 0,\ otherwise \end{aligned} \tag{3.9}$$

We refer to this scheme as *AllChains-Utility*.

tf.idf weighting

This is the classical *tf.idf* scheme used in the BoW model. Since this is the most popular feature weighting scheme for documents clustering, we feel it would be a good reference point for comparing the performance of the lexical chain based features. The *tf.idf* weight is a measure used to evaluate how important a word is to a document in a collection or corpus. The importance of a word feature increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus. The idea here is that if a word occurs in a large number of documents in a corpus, it looses its discriminative power and therefore should not be used a feature. The idf part of the *tf.idf* formula will reduce the weightage of those features which occur in too many documents in a corpus.

Adapting *tf.idf* to the lexical chains is straightforward and is computed as follows. The term frequency, tf of a lexical chain L, is the number of times f a lexical chain occurs in a document.

$$tf(L) = log(1 + f(L)) \tag{3.10}$$

The term frequency is usually damped to smoothen out its rate of growth. That is, if a term (lexical chain in our case) occurs three times in document A and six times in document B, it does not imply that document B is twice as important as document A. The $\log$ in the above equation is used to reduce the range of the function.

	Using All Chains	**Using Chain Selection**
Using Binary weights	AllChains-Binary	SelectedChain-Binary
Using Utility weights	AllChains-Utility	SelectedChains-Utility
Using tf.idf weights	AllChains-tfidf	

Table 3.1: Five Types of Lexical Chains based Feature Vectors

The document frequency (df) of a lexical chain L is the number of documents in which it appears. The inverse document frequency (idf) is the inverse of df.

$$idf(L) = 1/log(1 + df(L)) \tag{3.11}$$

Thus, the weight of the component corresponding to a lexical chain L is

$$tf.idf = tf(L).idf(L) \tag{3.12}$$

We refer to this scheme as *AllChains-tfidf*. Thus, we have five different varieties of feature vectors spread across two categories. These are summarised in Table 3.1

CHAPTER 4

Evaluation

We evaluated the performance of our lexical chain based feature vectors on two machine learning tasks - clustering and classification. Our evaluations were mainly directed towards the following goals:

1. Deciding the best feature weighting scheme for the lexical chain based features
2. Deciding if the Noun based candidate words are adequate for representing features
3. Showing that identity and synonymy relations alone are enough to compute the lexical chains
4. Comparing the performance between lexical chains based features and bag of words based features
5. Comparing the performance between lexical chains based features and features obtained through latent semantic analysis (LSA)

We used the bisecting *k*Means algorithm [24] to decide the first three goals. Based on the conclusion of the first two sets of experiment, we performed the comparison between the lexical chain based features and bag of words features using both the bisecting *k*Means algorithm and a Co-clustering [25] algorithm. The comparison with the LSA features was done using the bisecting *k*Means algorithm. The classification experiments were limited to comparing the performance between lexical chains and bag of words features and were performed using Support Vector Machines [26]. Both clustering and classification experiments were conducted on datasets derived from the 20 Newsgroups [9] dataset.

We describe our experimental setup and the results in the following two sections. Section 4.1 discusses the clustering experiments namely the datasets used, the clustering algorithms used, followed by the experiments and results. Section 4.2 describes the classification experiments and results.

4.1 Clustering Experiments

Document clustering was used to evaluate the performance of various aspects of the lexical chain based features. Five sets of experiments were conducted on the clustering task. We first evaluated the performance of the five different feature weighting schemes discussed in Sections 3.3 and 3.4. Subsequently we evaluated the relative performance of the three categories of candidate words discussed in Section 3.1. Section 4.1.9 shows that identity and synonymy relations are adequate to compute lexical chains. We then compare the lexical chain based document features against the classical bag of words and evaluate their performance on document clustering. Finally we compare the performance of our features with those generated by using the Latent Semantic Analysis (LSA) technique. LSA is a purely data driven approach which exploits co-occurrence information to identify the semantics contained in the documents. The LSA technique, to the best of our knowledge, is the only known competitor to our approach.

The bisecting *k*Means algorithm was used for the clustering task. Steinbach, *et. al.* [27] reported that the bisecting *k*Means algorithm performed better than regular *k*Means and other agglomerative algorithms for document clustering. This result led us to chose the bisecting *k*Means as the representative algorithm for the clustering experiments. All our clustering experiments were conducted using the bisection *k*Means, except for the comparison with BoW features where we used a co-clustering algorithm in addition to the bisecting *k*Means. This was done to study if there were any artifacts of the algorithm which were influencing our results. As we show later, our approach works consistently well with both algorithms.

This section is organised as follows: The next two subsections describe the two clustering algorithms used for the experiments. This is followed by a discussion on the datasets used and a description of the evaluation measures used. The experiments and their results are described after this. This section closes with a discussion of the results.

4.1.1 Bisecting *k*Means

The bisecting *k*Means algorithm [24] is an extension of the regular *k*Means algorithm. This algorithm, though partitional in nature, can be considered to be a hybrid of partitional and hierarchical clustering algorithms. Bisecting *k*Means algorithm intially considers all the data-points as a single cluster. In each iteration, the algorithm selects one of the existing clusters and splits it into two using the regular *k*Means algorithm. This process is continued until the required number (k) of clusters are obtained. Since in each iteration an existing cluster is split, the bisecting *k*Means can be used to create a hierarchy of clusters. If a record of the clusters formed in each iteration is maintained, hierarchical clusters can be obtained. The cluster selection in each iteration can be done in various ways:

- Choose the cluster with the largest size
- Choose the cluster with the highest sum of squared error (SSE)
- Choose the cluster with both large size and high SSE

Steinbach, *et. al.* [27], reported that there is no significant difference in the performance between the three cluster selection criteria. A listing of the bisecting *k*Means algorithm given in [24] is shown as Algorithm 2. This listing has been simplified for better readability.

One drawback of the bisecting *k*Means algorithm is that the clusters obtained are optimised only locally within a parent cluster and not globally with respect to all the data-points. A remedy for this suggested in [24] is to use the final set of centroids obtained from the bisecting

Algorithm 2 Bisecting kMeans Algorithm

```
1: Initialise the list of clusters to contain a single cluster with all the data points
2: repeat
3:    Remove a cluster from the list of clusters
4:    for i = 1 to number of trials do
5:       Bisect the selected cluster using the regular kMeans with k=2
6:       if Output clusters have lower SSE then
7:          Discard previous pair of clusters and retain the new pair
8:          Update bestSSE values
9:       else
10:         Discard pair of clusters
11:      end if
12:   end for
13:   Add the selected pair of clusters to the list of clusters
14: until list of clusters contains K clusters
```

*k*Means algorithm to initialise a regular *k*Means algorithm and run it on the dataset to get the final output.

The regular *k*Means algorithm is known to work well for small values of k. Since in the bisecting *k*Means algorithm, each iteration uses a regular *k*Means clustering with $k = 2$, the bisection step will result in clusters of rather good quality. The regular *k*Means algorithm is highly sensitive to initialisation. The bisecting *k*Means is less sensitive to initialisation conditions since the dataset is clustered into only two sub clusters in each iteration.

Steinbach, *et. al.* [27] performed a comparative evaluation of the performance of the Hierarchical Agglomerative Clustering (HAC) algorithms and partitional clustering algorithms for document clustering. Their experiments revealed that the bisecting *k*Means performed better than the other algorithms evaluated.

We used the implementation of the bisecting *k*Means algorithm in the Cluto toolkit [28] for our clustering experiments. The Cluto toolkit provides two variations of the bisecting *k*Means algorithm. One variant is as shown in Algorithm 2. The other variant uses the centroids obtained from the first variant to initialise a regular *k*Means algorithm and uses the output of this regular *k*Means run to get the final clustering result. We used the second variant of the bisect-

ing *k*Means algorithm (switch -clmethod=rbr) for our experiments. Cluto provides a number of options to customise the algorithm and the way it is run. We describe here the options used for the experiments. The values of feature weights in the feature vectors are sometimes scaled to within a range (say -1 to 1 or 0 to 1). However, we did not use any row or column scaling (switches -rowmodel=none, -colmodel=none). For the bisecting step we select the cluster whose bisection will achieve the maximum improvement in the overall clustering criterion (switch -cstype=best) and the number of bisecting trials was set to 10 (switch -ntrials=10). Each dataset was clustered 10 times with a different seed value (switch -seed=xxx). The seed values were obtained from the random number generator in the Linux BASH shell.

4.1.2 Co-Clustering

Traditionally, clustering algorithms work along a single dimension, *i.e.*, the documents (the clustering dimension) are clustered based on the similarity of the words (the other dimension) contained in them . Co-clustering is a clustering technique which clusters along both the dimensions, *i.e.*, documents and words. It exploits the duality between the documents and the words contained in the documents [29].

Formally, let $X \epsilon \Re^{d \times t}$ be a matrix whose rows corresponds to the documents and columns correspond to the terms contained in the documents. Each element x_{ij} corresponds to the element in the i^{th} row and j^{th} column of this matrix and gives the frequency with which the term corresponding to column j occurs in the document corresponding to row i. The co-clustering algorithm partitions this matrix into k row clusters and l column clusters. If I corresponds to the indices of the rows in a row cluster and J corresponds to the indices of the columns in a column cluster, then the sub-matrix determined by I and J is called a co-cluster. Each co-cluster is optimised with respect to an objective function. Numerous formulations including spectral analysis [30], information theoretic measure [29] and minimum sum-squared residue [31] have

been used to compute the objective function.

The Minimum Sum-Squared Residue Co-clustering (MSSR) algorithm [31] uses two measures to compute the homogeneity of a co-cluster. The first of these used the sum of the squared differences between each entry in the co-cluster and the mean of the co-cluster as the residue of an element x_{ij} in X.

$$h_{ij} = x_{ij} - x_{IJ} \tag{4.1}$$

where $x_{IJ} = \frac{\sum_{i \epsilon I, j \epsilon J} x_{ij}}{|I|.|J|}$ is the mean of all the entries in the co-cluster.

The second measure used the sum of the squared differences between each entry in the co-cluster and the corresponding row mean and the column mean with the co-cluster mean added to it to maintain symmetry and is given by

$$h_{ij} = x_{ij} - x_{iJ} - x_{Ij} + x_{IJ} \tag{4.2}$$

where $x_{iJ} = \frac{\sum_{j \epsilon J} x_{ij}}{|J|}$ is the mean of the entries in row i whose column indices are in J and $x_{Ij} = \frac{\sum i \epsilon I x_{ij}}{|I|}$ is the mean of the entries in column j whose row indices are in I.

The residue matrix $H = [h_{ij}]_{m \times n}$ is formed from the residues of each element of X using either of the two measures. They formulated the optimisation problem so as to minimise the total squared residue to obtain the following objective function.

$$||H||^2 = \sum_{IJ} \sum_{i \epsilon I,\ j \epsilon J} h_{ij}^2 \tag{4.3}$$

We used an implementation of the co-clustering algorithm available from [32]. This toolkit implements three co-clustering algorithms one of which is the minimum sum-squared residue co-clustering algorithms described above. We used this algorithm for our experiments (option

-A r). Since we know the number of true clusters (refer Table 4.3), the number of row clusters (-R xxx) was set to this number. The algorithm also requires the user to specify the number of column clusters. Since we do not have an idea of the true number of column clusters, we used a technique adapted from [29]. The algorithm is repeatedly run with the number of column clusters increasing in powers of 2 and starting from the first power of 2 and ending at the highest power less than or equal to the number of dimensions in the dataset. The clustering result which has the lowest value of the objective function is used as the final result.

4.1.3 Dataset

Unlike text classification, where large scale and well analysed datasets are available, document clustering has no such well defined datasets. One solution to the problem is to manually analyse the clusters formed, an approach we used for evaluating the soft clustering algorithm described in the next chapter. But manual analysis is not feasible for any large scale experiments.

A widely used dataset reported in the clustering literature is the 20 Newsgroups corpus. The 20 Newsgroups corpus is a collection of Usenet messages gathered from 20 Usenet groups. Each group contains between 700 to 1000 documents, to form a total of a little less than 19,000 documents. The 20 Usenet groups present in 20NG are listed in Table 4.1. Unlike other document datasets like Reuters [33, 34] which contain documents written mainly by trained journalists, the 20NG contain messages which are written by a wide population of net users. This naturally results in a good variation in grammar, choice of words and writing style. This we feel will be more representative of the real world data than a closed user group dataset such as Reuters.

The 20NG corpus can be considered to contain 20 clusters, each belonging to a different Usenet group. We are interested in finer granularity clusters and to the best of our knowledge, no such datasets exists for text. We worked around this problem by taking the documents in

Usenet Groups	
comp.graphics	sci.crypt
comp.os.ms-windows.misc	sci.electronics
comp.sys.ibm.pc.hardware	sci.med
comp.sys.mac.hardware	sci.space
comp.windows.x	talk.politics.misc
rec.autos	talk.politics.guns
rec.motorcycles	talk.politics.mideast
rec.sport.baseball	talk.religion.misc
rec.sport.hockey	soc.religion.Christian
alt.atheism	misc.forsale

Table 4.1: Usenet Groups contained in 20 Newsgroups dataset

5 Usenet groups and further grouped the documents on the basis of their subject line. We treated each subgroup thus formed as a cluster. Since, 20NG consists of email messages, documents with the same subject line[1] are assumed to be closely related and can be placed in the same cluster. This results in fairly tight clusters. An example of the clusters formed for the `alt.atheism` dataset is shown in Table 4.2. This grouping into classes is used as the gold standard for evaluating the clustering algorithms.

Cluster	Document Ids
college atheists	53675, 53357, 53540
amusing atheists and anarchists	53402, 53351
Islam & dress code for women	51212, 51216, 51318

Table 4.2: Example of the clusters obtained from grouping the documents using the subject line

We created five datasets from five distinct groups of the 20NG corpus - *comp.windows.x* (`cwx`), *rec.motorcycles* (`rm`), *talk.politics.guns* (`tpg`), *soc.religion.christian* (`src`) and *alt.atheism* (`aa`). The statistics for these datasets are given in Table 4.3.

We prepared the dataset for feature extraction by removing the complete header including the subject line and used only the body portion of the messages to compute the features. We extracted features on this data using both the Bag of Words (BoW) and lexical chains scheme.

[1] 'Re:', 'Fwd:', *etc.*, are stripped from the subject line before the grouping is performed

Collection	Code	# Clusters	# Documents
alt.atheism	aa	196	799
comp.windows.x	cwx	649	980
rec.motorcycles	rm	340	994
talk.politics.guns	tpg	267	910
soc.religion.christian	src	351	997

Table 4.3: Dataset Statistics

4.1.4 Evaluation Measure

Cluster evaluation can be done using two methods - intrinsic and extrinsic. The intrinsic methods measure some property of the clusters such as entropy or cluster purity. In the extrinsic method, the clustering result is embedded in a larger task such as information retrieval and the performance of the clustering results is measured on the basis of the improvement achieved for the larger task.

An edit distance based measure for measuring cluster quality was proposed by Pantel, *et. al.* [35]. They defined the edit distance as the number of merge, move and copy operations required to transform the resulting clusters to the gold standard. Initially, if there are c classes in the gold standard, c empty clusters are created. The measure then merges each resulting cluster to the cluster in the gold standard with which it has maximum overlap, breaking ties randomly. Thus, the merge operation attempts to bring the obtained clusters as close as possible to the gold standard as a whole. Subsequently, the move and copy operations are used to move or copy the documents around so that they finally match the gold standard. The number of the merge, copy and move operations required to transform one set of clusters to another is called the edit distance. Given a gold standard A, a baseline clustering B where each document is in its own cluster and a clustering C, they defined the quality of the clustering C as

$$1 - \frac{dist(C, A)}{dist(B, A)}$$

Cluster	Documents in Cluster
GC_1	D_1, D_9, D_{10}
GC_2	D_2, D_5, D_4, D_6
GC_3	D_3, D_7, D_8

Table 4.4: Example Gold Standard Clusters

where $dist(.,.)$ is the edit distance. This measure gives the savings obtained from using the clustering results to construct the answer key versus constructing it from the baseline.

We used a variant of the above scheme to evaluate the goodness of the clusters. We performed all our experiments with a fixed number 'k' of clusters which is the number of clusters identified in the gold standard. We observed that the merge operation would inevitably add as many clusters as there are in the gold standard to the final count, *i.e.*, simply add k to each edit distance, skewing the results. Hence, we define the edit distance as only the number of move and copy[2] operations required to convert the obtained clusters to that of the gold standard. In effect, our edit distance measures the number of documents which are misplaced with respect to the gold standard. This is in effect the objective of clustering algorithms - achieve a grouping which is as close as possible to the real grouping, and can be used to indicate the 'accuracy' of the clustering. The obtained edit distance can be normalised by dividing it with the number of documents in the dataset. This will normalise the value of the measure to range between 0 and 1 allowing us to compare the results across different datasets. The lower the value of this measure, the closer the obtained clustering is to the gold standard.

We illustrate our edit distance based measure with an example. Consider 10 documents $D_1, \cdots, D_{10}$ which form 3 clusters $GC_1, \cdots, GC_3$ as shown in Table 4.4. Let this set of clusters be our gold standard. We apply a clustering algorithm on these 10 documents and obtain 3 clusters $RC_1, \cdots, RC_3$ as shown in Table 4.5

The edit distance is computed as follows: We take each gold standard cluster and find the result

[2]The copy count will be included only in the case of overlapping clusters, which happens if a document is in more than one cluster.

Cluster	Documents in Cluster
RC_1	D_3, D_8, D_7, D_6
RC_2	D_2, D_5, D_9
RC_3	D_1, D_{10}, D_4

Table 4.5: Example Clustering Results

cluster with which it has maximum overlap. For example, the result cluster with maximum overlap with GC_1 is RC_3. To transform RC_3 to GC_1 we have to remove 1 document (D_4) from the cluster and add 1 document (D_9) to the cluster. This results in two move operations and hence an edit distance of 2. Similarly, RC_2 requires 1 document to be removed and 2 to be added to it to transform it to GC_2. This results in 3 moves and the total edit distance becomes 5. RC_1 requires 1 document to be removed to transform it to GC_3 and hence the total edit distance for this clustering result is 6. If we normalise it with the number of documents, we get a normalised edit distance of 0.6. In the best case scenario, no document will have to be moved resulting in an edit distance of 0, and in the worst case, all 10 documents have to be moved resulting in an edit distance of 10. The copy operation is required in the case of soft-clustering where documents will appear in multiple documents.

4.1.5 Experiments

In Sections 3.3 and 3.4 we proposed five feature weighting schemes which we felt would be useful as feature weights for lexical chain based features. We evaluated the relative performance of these five schemes on the clustering task using the bisecting *k*Means algorithm described in Section 4.1.1. We first computed the lexical chains for the five datasets described in Section 4.1.3 using all the three categories of candidate words (N, NV and NVA) described in Section 3.1 separately. This results in three different global sets (one each for N, NV and NVA) for each of the five datasets. Subsequently, we generated feature vectors, using the five varieties of feature weighting schemes, for each of the fifteen global sets. We refer to the five

Edit Distance			
	Candidate Words Categories		
	N	NV	NVA
SelectedChains-Binary	100 (0.125)	103 (0.129)	101 (0.126)
SelectedChains-Utility	77 (0.096)	**76 (0.095)**	80 (0.100)
AllChains-Binary	102 (0.128)	101 (0.126)	102 (0.128)
AllChains-Utility	**71 (0.089)**	78 (0.098)	**78 (0.098)**
AllChains-tfidf	106 (0.137)	102 (0.128)	101 (0.126)

Table 4.6: Edit distances for the **alt.atheism** dataset

Edit Distance			
	Candidate Words Categories		
	N	NV	NVA
SelectedChains-Binary	283 (0.289)	291 (0.297)	291 (0.297)
SelectedChains-Utility	**207 (0.211)**	**212 (0.216)**	**207 (0.211)**
AllChains-Binary	304 (0.310)	295 (0.301)	300 (0.306)
AllChains-Utility	217 (0.221)	228 (0.233)	219 (0.223)
AllChains-tfidf	288 (0.294)	293 (0.299)	292 (0.297)

Table 4.7: Edit distances for the **comp.windows.x** dataset

varieties of feature weighting schemes as SelectedChains-Binary and SelectedChains-Utility for the binary weighted and utility weighted schemes described in Section 3.3 and AllChain-Binary, AllChains-Utility, AllChains-tfidf for the binary, utility and tf.idf weighted schemes described in Section 3.4. These 75 sets of feature vectors were clustered using the bisecting *k*Means algorithm described earlier.

The edit distances of the resulting clusters were computed as described in Section 4.1.4 and the best edit distance obtained out of 10 runs of clustering algorithm was taken. The results, for each set of feature vectors, are enumerated in Tables 4.6- 4.10. The values in bold are the best edit distance values in each column and the bracketed values are the normalised edit distances.

Edit Distance			
	Candidate Words Categories		
	N	NV	NVA
SelectedChains-Binary	105 (0.115)	108 (0.118)	111 (0.122)
SelectedChains-Utility	**61 (0.067)**	**72 (0.079)**	**74 (0.081)**
AllChains-Binary	125 (0.137)	128 (0.141)	121 (0.133)
AllChains-Utility	64 (0.07)	78 (0.0857)	75 (0.0824)
AllChains-tfidf	114 (0.125)	123 (0.135)	116 (0.127)

Table 4.8: Edit distances for the **talk.politics.guns** dataset

Edit Distance			
	Candidate Words Categories		
	N	NV	NVA
SelectedChains-Binary	187 (0.188)	187 (0.188)	197 (0.198)
SelectedChains-Utility	134 (0.134)	130 (0.130)	130 (0.130)
AllChains-Binary	207 (0.208)	209 (0.210)	210 (0.211)
AllChains-Utility	**131 (0.131)**	**129 (0.129)**	**126 (0.126)**
AllChains-tfidf	209 (0.210)	212 (0.213)	212 (0.213)

Table 4.9: Edit distances for the **soc.religion.christian** dataset

Edit Distance			
	Candidate Words Categories		
	N	NV	NVA
SelectedChains-Binary	198 (0.199)	202 (0.203)	203 (0.204)
SelectedChains-Utility	119 (0.119)	**128 (0.129)**	**129 (0.129)**
AllChains-Binary	210 (0.211)	206 (0.207)	204 (0.205)
AllChains-Utility	**118 (0.119)**	134 (0.135)	132 (0.133)
AllChains-tfidf	209 (0.210)	198 (0.199)	205 (0.206)

Table 4.10: Edit distances for the **rec.motorcycles** dataset

4.1.6 Results

The results show that both the binary weighted schemes and tf-idf scheme performs worse than the utility weighted schemes. This validates our hypothesis that the value of $util(.)$ function gives a better weighting for the features. The binary weighting scheme gives uniform weights to the features which could explain its lower performance. The $util(.)$ computes the utility of each feature to the document as a function of both the strength of the chain and the presence of the chain in a document, thereby giving a better indication of the weight of each feature in the vector.

The results in Tables 4.6- 4.10 indicate that the SelectedChains-Utility is the best for two (`tpg` and `cwx`) of the five datasets, while the AllChains-Utility is best for `src` dataset. The results are split between SelectedChains-Utility and AllChains-Utility for the other two datasets (`aa` and `rm`). While the results are conclusively in favour of the utility based feature weighting, the choice between SelectedChains and AllChains is a little more difficult. As mentioned earlier, our objective is to obtain a representation with the minimum number of features which does not degrade the performance. The SelectedChains scheme results in a smaller number of features as compared to the AllChains scheme. Thus, if we could show that the performance between the two schemes is nearly the same, we could then select the SelectedChains scheme and take advantage of the smaller number of features. Towards this end, we performed a statistical test to help us make the decision.

4.1.7 Evaluation of Feature Weighting Schemes

We analysed the performance between the two feature weighting schemes by the Bootstrap Method [36]. The bootstrap method is a well known statistical technique used to perform analysis on sample statistics. In our case the statistic is the edit distance. The bootstrap is particularly useful when the amount of data available is limited. The bootstrap method works

	SelectedChains		AllChains		95% CI	P value
	Mean	SD	Mean	SD		
aa	86.40	8.66	85.60	6.45	[-6.55,8.15]	0.8110
cwx	263.70	11.27	263.00	17.93	[-11.11,12.51]	0.8963
rm	152.70	4.88	153.11	10.19	[-7.55,4.89]	0.6344
src	148.00	10.26	146.00	7.70	[-5.13,8.91]	0.5521
tpg	91.60	10.21	90.80	6.61	[-9.45, 11.05]	0.8638

Table 4.11: Results of the bootstrap analysis

by creating new 'bootstrap datasets' from the original dataset by sampling from it with replacement. The sample statistic is then computed from each of these bootstrap datasets and used for further analysis such as t-tests [37] or ANNOVA [38]. In our case it will be sufficient to show that the difference in performance between the SelectedChains-Utility and AllChains-Utility is not statistically significant. This would then allow us to conclusively select the SelectedChains-Utility scheme.

Towards this end, we created 10 bootstrap datasets for each of the five datasets and the two feature weighting schemes - AllChains and SelectedChains. These bootstrap datasets were then clustered and the cluster quality was measured in terms of the edit distance. A paired t-test was performed on the edit distances to check if differences in the means of the edit distance were statistically significant. In our analysis, we used 10 datasets per group and compared the performance between SelectedChains and AllChains for each of the datasets. The results are enumerated in Table 4.11. The table lists the mean and standard deviation of the edit distances along with a 95% confidence interval for the difference in the means and the P value obtained for the t-test.

Based on the P value reported in Table 4.11, we can conclude that there is no statistically significant difference in the means of the edit distances. In other words, the performance of the SelectedChains scheme is nearly the same as that of the AllChains scheme. Moreover, the dimensionality of the generated feature vectors listed in Table 4.12 shows that the SelectedChains feature vectors are smaller than AllChains by a factor of 6, on an average. Since the

	Feature Vector Dimensions with	
	SelectedChains	AllChains
alt.atheism	860	6009
comp.windows.x	671	4582
rec.motorcycles	918	5310
soc.religion.christian	713	6850
talk.politics.guns	929	6316

Table 4.12: Dimensionality of the Feature Vectors for N category Candidate Words

SelectedChains scheme results in a lower number of dimensions, we can conclusively select the SelectedChains-Utility scheme as the best option.

4.1.8 Evaluation of Candidate Word Categories

In the previous section, we showed that the SelectedChains-Utility feature vector representation is the better choice among the five. The choice between the three categories of candidate words is more difficult. All the three categories give comparable performance. But it will be advantageous to prefer the category of candidate words which results in a smaller number of features and which does not degrade the performance significantly. The dimensionality of the feature vectors formed across the three categories for the SelectedChains-Utility feature vectors for all the datasets is shown in Table 4.13.

The NV category increases the dimensionality by at least 19% over N, and the NVA category increases it by at least 41% over N. The variation in performance between the three categories is not that significant as is evident from Tables 4.6- 4.10. This implies that using only the N category of candidate words captures all the required information to successfully discriminate the documents. We can also take advantage of the smaller number of features when only the N category is used.

We once again use the bootstrap method described in the previous section to conclusively show that there is a no statistically significant difference in the performance between the three

	Feature Vector Dimensions				
	N	NV	Increase over N	NVA	Increase over N
alt.atheism	860	1056	23%	1297	51%
comp.windows.x	671	826	23%	959	43%
rec.motorcycles	918	1095	19%	1291	41%
soc.religion.christian	713	940	31%	1162	63%
talk.politics.guns	929	1129	22%	1356	46%

Table 4.13: Dimensionality of the Feature vectors for the 3 categories of Candidate Words in the SelectedChains case

	N		NV		95% CI	P value
	Mean	SD	Mean	SD		
aa	86.40	8.66	84.89	4.94	[-7.30, 7.30]	1.00
cwx	263.70	11.27	271.40	16.11	[-23.22, 7.82]	0.2909
rm	152.70	4.88	151.33	7.18	[-4.55, 5.44]	0.8426
src	148.00	10.26	147.10	9.33	[-10.35,11.69]	0.8925
tpg	91.60	10.21	92.10	5.86	[-9.99, 8.99]	0.9078

Table 4.14: Bootstrap Analysis of N vs. NV candidate words

categories of candidate words. We used the datasets corresponding to the SelectedChains-Utility scheme to generate 10 bootstrap datasets by sampling with replacement from each of the five datasets. We compared the performance of N *vs.* NV and N *vs.* NVA for each dataset. Each of the bootstrap datasets were clustered and the edit distances were computed. The computed edit distances were subjected to a paired t-test to check if the difference in means of the edit distances was statistically significant. The results are enumerated in Tables 4.14 and 4.15. The table lists the mean and standard deviation of the edit distances along with a 95% confidence interval for the difference in the means and the P value obtained for the t-test.

Based on the P values obtained, the tests revealed that the difference in performance between the N and NV category and the N and NVA category of candidate words is not statistically significant. We can thus conclude that the N category of Candidate words is adequate to represent the documents. This to the best of our knowledge, is also the first attempt in the lexical chain literature to study the performance of non-noun categories of candidate words.

	N		NVA		95% CI	P value
	Mean	SD	Mean	SD		
aa	86.40	8.66	82.22	6.22	[-4.56,9.90]	0.4198
cwx	263.70	11.27	265.20	16.14	[-17.06,14.06]	0.8322
rm	152.70	4.88	147.22	12.58	[-5.02, 14.13]	0.3047
src	148.00	10.26	147.44	8.83	[-9.68,10.79]	0.9035
tpg	91.60	10.21	92.10	10.05	[-11.60,10.60]	0.9211

Table 4.15: Bootstrap Analysis of N vs. NVA candidate words

4.1.9 Lexical Chains using Other Relations

In the previous chapter, we hypothesised that using only the identity and synonymy relations to form the lexical chains will result in chains which represent crisp topics, each of which contains a well defined concept. We had argued that using other relations will result in chains which represent composite but possibly meaningless concepts.

We performed a set of experiments to test this hypothesis. We computed the lexical chains as given in the previous chapter, but used hypernymy and co-ordinate terms (Section 2.4) in addition to the identity and synonymy relations to compute the chains. The hypernymy relation is a generalisation of a concept while the co-ordinate terms gives other terms that share a hypernym. These two WordNet relations together would be able to create chains which contain composite topics of just the right granularity. That is, ‘car’ and ‘motorcycle’ might be put in a single chain, but not ‘snow’ and ‘car’.

Since this could result in candidate words which could belong to multiple chains, we used a decision scheme where we first search for a chain with which it has an identity or synonymy relation. If no such chains are identified, we search for chains with which it has a hypernymy relationship. Failing this we look for chains with which it has a co-ordinate term relationship. If such a chain is found, we add it to the chain, or else we create a new chain. This scheme is very similar to the other algorithms reported previously in the literature. (Refer to Section 2.2). We used the SelectedChains-Utility option for chain selection and feature weighting and used

	LC-OR	LC
alt.atheism	89	77
comp.windows.x	212	207
rec.motorcycles	125	119
soc.religion.christian	144	134
talk.politics.guns	70	61

Table 4.16: Edit distance obtained from clustering using a modified lexical chain computing algorithm. Here LC-OR refers to Lexical Chains computed using Other Relations.

only candidate words belonging to the N category. We used an experimental setup identical to the previous experiments, with the bisecting *k*Means algorithm as the clustering algorithm. The best edit distance values obtained from 10 clustering runs are shown in Table 4.16, where LC-OR refers to lexical chains computed using other relations. The corresponding edit distances (column 'LC') from the experiments using only the identify and synonymy relations are reproduced for comparison.

The results for the LC-OR case are similar to the baseline LC case. An analysis of the lexical chains generated revealed the cause for this result. While the lexical chains generated are in line with our algorithm, the chain selection process actually ends up selecting nearly the same chains as the baseline case. This is due to the fact that only a small number of chains from the baseline case are affected by the modified algorithm. This leads us to conclude that the benefits derived from using the modified algorithm is limited and therefore can be ignored.

4.1.10 Lexical Chain *vs.* Bag of Words based Features

The previous two experiments showed that the SelectedChains-Utility along with the N category candidate words are the best choices among the alternatives. We now shift our focus towards evaluating the utility of representing documents using the lexical chains (*lexchains*) scheme over the BoW. We extracted features on each of the datasets using both the BoW and lexical chains scheme. For the BoW scheme, we first tokenised the document, filtered out the

	Feature Vector Dimensionality		
	BoW	lexical chain	Approx. Reduction
aa	8881	860	10x
cwx	12767	671	19x
rm	8675	918	9x
src	11115	713	15x
tpg	10755	929	11x

Table 4.17: Dimensionality of the Feature Vectors

stopwords using the list obtained from [39] and further stemmed them using a Porter Stemmer [4]. The feature vectors were then computed using the *tf.idf* scheme. We refer to the feature vectors thus obtained as cwx-BoW, rm-BoW, tpg-BoW, src-BoW and aa-BoW for the cwx, rm, tpg, src and aa datasets respectively (Section 4.1.3). Lexical chain based features were derived using the SelectedChains-Utility scheme described in Sections 3.3 and are analogously referred to here as cwx-lc, rm-lc, tpg-lc, src-lc and aa-lc. This results in a total of ten datasets.

The dimensions of the feature vectors obtained are summarised in Table 4.17. It can be noted that the size of the lexical chains based features reduces the dimensions of the feature vectors by at least 9 times as compared to the bag of words representation. This, as we show later, results in a significant improvement in the run time of the algorithms.

The ten datasets were clustered using the bisecting *k*Means algorithm. The value of k was set to the true number of document clusters *i.e.*, *k* was set to 649, 340, 267, 351 and 196 for cwx, rm, tpg, src and aa respectively. This reflects the number of clusters in the gold standard (*ref.* Table 4.3).

The results are enumerated in Table 4.18. The lexical chain based document features results in a significant improvement in performance over the BoW representation. The lexical chains based document features gives an improvement of upto 33% over the BoW representation while achieving a reduction in dimension of the feature vectors by more than 9 times.

	Edit Distance	
	*k*Means	CoCluster
aa-BoW	102 (0.128)	92 (0.115)
aa-lc	**77 (0.096)**	**73 (0.091)**
cwx-BoW	292 (0.297)	221 (0.225)
cwx-lc	**207 (0.211)**	**217 (0.221)**
rm-BoW	192 (0.193)	195 (0.196)
rm-lc	**119 (0.12)**	**123 (0.124)**
src-BoW	201 (0.202)	183 (0.229)
src-lc	**134 (0.134)**	**134 (168)**
tpg-BoW	131 (0.144)	110 (0.121)
tpg-lc	**61 (0.067)**	**100 (0.11)**

Table 4.18: Comparison between Lexical Chains and Bag of Words based features on bisection *k*Means and Co-clustering. Normalised edit distances are given in parenthesis

One of the advantages of the reduced dimensionality is the shorter run-time taken by the algorithms to converge. We verified this by performing run time studies on the dataset. We used the co-clustering toolkit for these experiments. We do not include the time taken to compute the features for two reasons. First, the run time required to compute the BoW and lexical chain based features are implementation dependent and therefore, the runtimes cannot be directly compared as an objective benchmark. Secondly, we observed that the time required to compute both the BoW and lexical chain features are nearly the same with the latter taking slightly longer than the former. We believe that this is an implementation issue and can be rectified with a tighter implementation. The results are the run-times reported by the clustering tool used and are averaged over four runs. The average time taken by the clustering algorithm to converge for each of the datasets are enumerated in Table. 4.19.

The results show that the lexical chain based features produced a speedup of atleast 50% as compared to the bag of words based features. Thus the use of lexical chains based features results in a significant improvement in the run-time performance of the algorithms while maintaining or improving the clustering performance.

	Time in secs	
	BoW	Lexical Chains
aa	64.824	41.079
cwx	233.043	76.293
rm	106.087	59.736
src	134.356	80.001
tpg	92.698	54.271

Table 4.19: Runtime performance on the clustering task

4.1.11 Lexical Chains *vs*. LSA features

The previous subsection conclusively showed that the Lexical Chains based feature vectors outperform the bag of words features. We felt that to conclusively prove the effectiveness of our scheme it was imperative to compare it against a more 'semantically' oriented feature representation scheme. The Latent Semantic Analysis (LSA) [40] is a well known method for generating such features.

LSA is a technique developed to induce and represent the meaning hidden in the text. LSA can be viewed as an enhancement to the vector space model. LSA uses a smaller dimensional approximation to a larger dimensional relationship between terms and documents. This reduced dimensional space is hypothesised to represent a conceptual space where documents are represented in terms of the concepts contained in them. This idea is similar to our lexical chains based document representation mechanism. The key difference between the LSA technique and our technique is that LSA is a purely data driven technique, while ours is a hybrid of both data and knowledge driven techniques.

4.1.11.1 Latent Semantic Analysis

LSA was proposed as a technique to overcome the problem of lack of semantics in the BoW representation for information retrieval tasks. LSA uses a linear algebra technique known as Singular Value Decomposition (SVD) [41] to transform the original 'n' dimension representa-

tion in the term space to a smaller 'k' dimensional representation in the concept space.

LSA works on the premise that a concept can be represented using multiple terms. Such terms co-occur in a passage or text document. This co-occurrence information is obtained from a term-document matrix. The SVD analysis uncovers the major co-occurrence patterns while ignoring the smaller and less significant ones. Thus terms which do not appear in a document may get some weightage because it co-occurs with other terms present in other documents.

The LSA takes a term-document (t-d) matrix as its inputs. The t-d matrix is a rectangular matrix with the terms represented as the rows of the matrix and the documents as the columns of the matrix. Thus each cell of this matrix gives the frequency of a term in a document. A single row in the matrix will give the frequency of a term in each of the documents, while a column will give the frequency of the different terms in a document.

Singular Value Decomposition (SVD) is a form of factor analysis. It constructs an n dimensional abstract semantic space in which each original term and each original document are represented as vectors. Using SVD, the rectangular t-d matrix X is decomposed into the product of three matrices U, S, and V' such that $X = U.S.V'$.

U is a orthonormal matrix and its rows correspond to the rows of X, but has m columns corresponding to new, specially derived variables such that there is no correlation between any two columns; *i.e.*, each is linearly independent of the others. V is an orthonormal matrix and has columns corresponding to the original columns but m rows composed of derived singular vectors. The third matrix S is an m by m diagonal matrix. The non-zero entries in this matrix are known as singular values. A large singular value indicates a large effect of this dimension on the sum squared error of the approximation. The role of these singular values is to relate the scale of the factors in the other two matrices to each other such that when the three components are matrix multiplied, the original matrix is reconstructed.

Following the decomposition by SVD, the k highest singular values in S are selected. These

are considered to be the most important dimensions for the input data. All other dimensions are omitted, *i.e.*, the other singular values in the diagonal matrix along with the corresponding singular vectors of the other two matrices are deleted. The choice of k decides the amount of dimensionality reduction. Ideally, k should be large enough to fit the latent structure in the data, but small enough such that noise, sampling errors or unimportant details are not modeled [40].

Deerwester *et. al.*, [40] using a test database of medical abstracts, showed that LSA performance improves considerably after 10 or 20 dimensions, peaks between 70 and 100 dimensions, and then begins to diminish slowly. Eventually, performance must approach the level of performance attained by standard vector methods, since with $k = n$ factors it will exactly reconstruct the original term by document matrix, X.

The reduced matrix ideally represents the important and reliable patterns underlying in the data in X. It corresponds to a least-squares best approximation to the original matrix X. The reduced matrix can be considered to be a vector space in k dimension. The documents are then represented in this reduced dimensional space. Each dimension in turn can be considered to be a semantic concept, which in effect is a combination of terms which co-occur.

Each dimension in this lower dimensional space may be a combination of one or more terms contained in the higher dimensional space. The intuition behind this is that each dimension in the reduced space represents a 'concept' as against the more atomic term. Thus, LSA can be considered to be a technique which transforms raw co-occurrence information between terms and documents into a conceptual space. This is based on the observation that synonymous words co-occur and that if such co-occurrence terms can be identified and mapped to a single dimension, we can obtain a mapping into concept space.

The LSA is a technique which is a purely data driven form of what we achieve using lexical chains. While it has the advantage that it does not depend on external databases it suffers from at least two drawbacks. One, since it is purely data driven, it could result in concepts that are

	Edit Distance								
	LSA								Lexical Chains
	k=10	k=15	k=50	k=100	k=150	k=200	k=300	k=400	
aa	95	**91**	94	98	97	95	101	102	**77**
cwx	307	302	298	**280**	282	294	298	303	**207**
rm	185	176	**170**	182	172	182	191	184	**119**
src	178	188	**176**	186	188	199	202	203	**134**
tpg	112	**95**	105	106	126	120	124	130	**61**

Table 4.20: Comparison between Lexical Chains and LSA based features

quite meaningless, *i.e.*, the terms car, flower and driver could be combined to a single concept simply because these three terms co-occur frequently in the given corpus. But the concept itself has little basis in the real world. The second problem arises from the computational cost of doing LSA. The SVD is a computationally intense algorithm and usually can take many minutes or even hours to compute. This could be cumbersome for large datasets. Infact, when we conducted our experiments using Matlab, we were unable to obtain a full decomposition of our t-d matrix. Instead, we had to resort to a light weight SVD algorithm which computed the decomposition for only the top k singular values. Even then we were limited to a k of around 600 only!

We used Matlab to compute the features using the LSA technique. We used the *svds* command in Matlab to decompose a t-d matrix. We used the *svds* command instead of the standard *svd* command because the latter was not able to decompose original t-d matrix we passed as input to it because of its size. We used the bag of words features as the starting point. These feature vectors were assembled into a t-d matrix which was then passed to the *svds* command. Apart from the t-d matrix, the *svds* command requires the user to specify the value of k which specifies the number of largest singular values to compute. The decomposed matrices are rank reduced and multiplied to obtain the LSA feature matrix. The feature vectors obtained were clustered using the bisecting *k*Means algorithm with k set to the same values as those used in the previous experiments. The results are enumerated in Table 4.20

As mentioned earlier, the best performance using the LSA scheme is observed between $k = 15$ and $k = 100$. Nevertheless, our lexical chain based approach gives better edit distances as compared to the LSA approach. It must also be mentioned that the LSA is a computationally intense technique. Since we used Matlab to perform the SVD analysis, a direct comparison of run times is not fair. But, we must point out that the lite version (*svds*) of SVD command in Matlab took a few hours to compute, while the Lexical Chain and Bag-of-Words based features were computed in order of minutes.

4.1.12 Discussion

Clustering algorithms seek to group the documents based on their semantic content. A document is not just a bunch of loose words as treated by the bag of words representation. Each word in a document contributes to some aspect of the overall semantics of the document. The BoW scheme inherently throws away a lot of information, which would have otherwise been useful in discerning the semantics of the document. The BoW representation fails to capture and represent these semantics resulting in a less accurate representation for the documents. This fact is reflected by the higher edit distance in the case of BoW based clustering in Table 4.18.

Currently, 'bag of words' representation is the *state-of-the-art* and is the most preferred and commonly used scheme for extracting features from documents. The method obviously ignores any semantic characteristics of text documents. Our belief that linguistically and semantically motivated features can play a very critical role in better representation of documents is the main motivation for this work. Previously, Hatzivassiloglou, *et. al.* [42] had studied the effects of linguistically motivated features on clustering algorithms. They had explored two linguistically motivated features - noun phrase heads and proper names and compared these against the BoW representation. They had reported that the BoW representation was better than linguistically motivated features. However, as reported in [42] Noun phrase heads and proper nouns alone

are inadequate representations. This is because these are just two simple aspects of the many possible aspects and characteristics contained in the text. These, therefore, are not capable of modelling all the intricacies of human language. Consequently, a more composite representation is required to obtain better results on semantically oriented tasks.

Lexical chains appear to be capable of doing this to a certain extent. During the process of computing and selecting the lexical chains, we are implicitly trying to decode the semantics of the documents. Lexical chains work on the basic premise that a document describes topics through a combination of words and these words will exhibit a cohesion among them. This cohesion can be identified using a resource such as WordNet [8]. In the process, lexical chains capture some amount of the semantics contained in the documents, resulting in a better performance in subsequent processing of the documents.

This has been conclusively demonstrated through our experiments where we compared features extracted using lexical chains against features using the bag of words representation. The lexical chains based features are lower in dimension, yet exhibit much better performance.

4.2 Classification Experiments

The experiments discussed in the previous section have demonstrated convincingly that lexical chain based features result in both accuracy and run time improvements for document clustering. We now consider the effects of lexical chain based features on document classification.

Document classification is the process of placing documents into one or more categories from a set of predefined categories. Unlike clustering, which is an unsupervised learning technique, classification is a supervised technique. The 'supervision' is in the form of a training set of labelled examples which gives the assignment of documents to categories. The learning algorithm then learns a classifier which automatically assigns the documents to each of the

labels.

As in clustering, documents are represented using a set of features. Traditionally, the bag of words representation is the most commonly used representation. The individual features can be weighted using many schemes such as binary, term frequency, and tf.idf. As with clustering, the feature vectors obtained using the bag of words approach results in a large number of dimensions. It will be advantageous to see if the lexical chains based feature vectors results in better performance for document classification tasks.

Numerous algorithm have been proposed for document classification, and these have been successful to various extents. The application of Support Vector Machines (SVM) [43] to text classification was pioneered by Joachims [44] who showed that they are particularly good for text classification [44]. Hence it is the algorithm of choice for our classification experiments. Since we are interested in a straight-forward performance evaluation betwen the two feature representation schemes for text classification, we followed a very standard procedure with established tools and techniques without any improvisations. This was done to achieve a head-to-head comparison of the performance of lexical chain based features against the BoW feature without any influence of the experimentation methodology used. The experiments described in the following sections were performed using the libSVM toolkit [45] .

4.2.1 Support Vector Machines

Support Vector Machines (SVMs) were proposed by Vapnik, *et. al.* [46], as a class of linear classifiers. The method works by finding a linear separating hyperplane that maximally separates out the data into two categories. SVMs preprocess the input data to represent them in a high dimensional space. This is based on the idea that any set of input data will be linearly separable by a hyperplane if they are projected onto a sufficiently high dimensional space. In effect, SVM assumes that the input data is transformed to a high dimensional space $z_i = \Phi(x_i)$. The

function $\Phi(.)$ is referred to as a kernel function. This kernel function can be a custom function derived based on the characteristics of the problem domain or can be one of the many known general kernel functions such as polynomial, gaussian, rbf, *etc*. Thus SVMs can be made into a non-linear classifier by using a kernel function.

As already mentioned, the objective of training a SVM is to find a separating hyperplane which separates out the data into two classes with the largest margin. This is based on the assumption that the generalisation performance of the classifier is improved if the margin is large. This optimal separating hyperplane is defined by 'support vectors' which are training patterns, that are equidistant from the separating hyperplane.

Given l training vectors $x_i \, \epsilon \, \Re^n$, $i = 1, .., l$, belonging to one of the two classes $\{-1, \, 1\} \, \epsilon \, Y$ and a corresponding vector of labels $y \, \epsilon \, \Re^l$ such that each $y_i \, \epsilon \, Y$ maps the corresponding x_i onto $\{-1, \, 1\}$. We want a maximally separating hyperplane which divides the training vectors into those belonging to class 1 and class -1. Given this setting, the SVM training can be cast as a quadratic optimisation problem. In the primal form, the optimisation problem is

$$\min \, \frac{1}{2} w^t.w$$

subject to

$$y_i(w^T \phi(x_i) - b) \geq 1$$

where w is the weight vector normal to the optimal hyperplane.

The primal form can be written in its dual form as

$$max \, \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \phi(x_i^T) \phi(x_i)$$

subject to

$$\alpha_i \geq 0$$

and

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

where, $\alpha_i \geq 0$ are the undetermined lagrangian multipliers. It can be seen from the dual formulation that the maximum margin hyperplane is a function of the support vectors.

We used the libSVM toolkit [45] for our experiments. This is one of the popular reference implementations of SVMs. This toolkit implements different SVM formulations. We used the C-SVM variant [47] for our experiments. We performed our experiments using both a linear kernel and a rbf kernel.

The performance of the SVM algorithm is highly dependant on the parameters used. The SVM using the rbf kernel required two parameters C and γ. Hsu, *et. al.*, [48] have suggested an exhaustive parameter search technique which they call as 'cross validation via parallel grid search'. In this approach, the training data is separated to several folds or partitions. Each fold is sequentially treated as a validation set and the remaining folds are used for training the SVM. The average of the accuracy for predicting each of the validation sets is the *cross validation accuracy*. Users provide a possible interval of C and γ. The SVM is then trained on all possible pairs of points C and γ within the given interval. The grid point which gives the highest cross validation accuracy is treated as the best parameter for training the SVM using the given data. We used the parameter identified through this process to train the whole training set and generate the final model.

4.2.2 Dataset

We used the 20 Newsgroups dataset for the classification experiments as well. We created three two-class problems by combining the documents from four groups into three pairs - we combined the alt.atheism and rec.motorcycles groups and the resulting dataset is referred to

Dataset	Class #1	Class #2	Total Documents
aa-rm	alt.atheism	rec.motorcycles.	1793
src-tpg	soc.religion.christian	talk.politics.guns	1907
rm-tpg	rec.motorcycles	talk.politics.guns	1904

Table 4.21: Dataset Statistics

as aa-rm. Similarly, the soc.religion.christian and talk.politics.guns were combined to create the src-tpg dataset and rec.motorcycles and talk.politics.guns were combined to get the rm-tpg dataset. The statistics of these three the datasets are listed in Table 4.21.

The BoW features were extracted as described for the clustering experiments. The lexical chain based features were also generated similarly. We used only the SelectedChains-utility feature weighting scheme on the N category candidate words. This scheme was the best performing scheme for clustering and therefore, we use it for the classification experiments as well. This resulted in six datasets - 3 each for BoW and LCs which were used for the classification experiments.

4.2.3 Experiments

We performed a 10 fold cross validation run on the datasets using the parameters identified using the grid search method described earlier. In this, the dataset is split into ten parts. One of the parts is set aside and the classifier is trained using the remaining nine parts. Once the classifier is trained, the accuracy is tested on the part set aside. This process is repeated for each of the ten parts, *i.e.* each of the ten parts is set aside and the classifier is trained on the remaining nine parts. The 10 fold cross validation will result in the SVM being tested on all the datapoints giving a better estimate of the performance than a single run with a training-test split. The average accuracy on the 10 fold cross validation is reported in Tables 4.22 and 4.23 corresponding to the RBF kernel and linear kernel respectively.

The results show that the performance of the lexical chains based features are not very

	Lexical Chains	BoW
aa-rm	96.97%	96.37%
src-tpg	97.57%	98.12%
rm-tpg	98.15%	98.16%

Table 4.22: Classification accuracy using the RBF kernel

	Lexical Chains	BoW
aa-rm	94.97%	98.32%
src-tpg	96.57%	98.85%
rm-tpg	94.97%	98.95%

Table 4.23: Classification accuracy using the linear kernel

different from that of the BoW features. While the classification results are not as exciting as the clustering results it does provides some insight. Firstly, since SVMs are capable of handling very high dimensional feature spaces, the impact of our approach on the performance of SVMs are limited. Secondly, the results we obtained for the bag of words based features are in line with the best results reported in the literature [44], which in itself is very high. Thus the marginal improvement which can be expected is limited.

We would also like to point out that training the classifiers using the cross-validation grid search approach on the lexical chains based features took just a few hours (3-4 hrs approx.) for each dataset, while the BoW words dataset took nearly 72 hours[3]. This practical advantage obtained by the speedup can easily be related to by anyone who has dealt with real and large datasets.

[3]The experiments were performed on a 64-bit AMD Athlon X2 machine with 1.25GB of RAM.

CHAPTER 5

Soft Clustering

Documents usually contain multiple strands of information. This multiplicity of topics is not captured when documents are clustered using a hard clustering algorithm. Hard clustering algorithms place each document in a single cluster. Each cluster will then be an amalgamation of multiple topic strands and will obfuscate the actual diversity of topics contained in the document collection. We feel it is more natural to cluster documents using a soft clustering algorithm, where a document will be placed in multiple clusters, possibly with varying degrees of membership. Each cluster will be more granular and will better reflect the distribution of topics in the collection.

The nature of our lexical chain based document representation makes it suitable for soft clustering documents. We propose here a soft clustering algorithm for documents using lexical chains. It is based on the premise that each lexical chain can be considered to represent a topical strand. This implies that, if two documents share a lexical chain, then they share a topic (or more precisely, a topic strand). Thus, if we can identify and group all documents which share a certain number (above a threshold) of lexical chains, we would obtain a clustering. Since, by this principle, a document being present in one cluster does not preclude it from being present in another cluster, we would obtain a soft clustering solution.

5.1 A Soft Clustering Algorithm

Consider a set of n documents $D = \{d_1, \cdots, d_n\}$. The soft clustering problem is to make an optimal assignment of each $d_i \, \epsilon \, D$ to a subset C' of clusters from $C = C_i, \cdots C_m$. We present

Algorithm 3 A Soft Clustering Algorithm using Lexical Chains

```
1:  for each document d do
2:      bool singleton = true
3:      for each existing cluster C_i in C do
4:          bool Add = true
5:          for each document x in C_i do
6:              if similarity(d,x) ≤ threshold then
7:                  Add = false;
8:                  break;
9:              end if
10:         end for
11:         if Add = true then
12:             Add document d to cluster C_i
13:             singleton = false
14:         end if
15:     end for
16:     if singleton = true then
17:         Create a new cluster and add cluster to C
18:         Add document to the new cluster
19:     end if
20: end for
```

here a simple algorithm for doing this which has been shown as Algorithm 3.

Each document is represented by a set of lexical chains. These lexical chains are computed in the same manner as described in Chapter 3. We apply the chain selection technique described in Section 3.3 to select a subset of lexical chains which best represent the topics contained in the document.

We maintain a set of clusters C initialised to a null set. For each input document d_i, the algorithm computes its similarity with each cluster $C_i \, \epsilon \, C$. The similarity measure used is the Dice Co-efficient [39]. The Dice Co-efficient is defined as

$$S(A, B) = \frac{2|A \cap B|}{|A| + |B|}$$

where $|x|$ is the number of lexical chains in document x. The numerator is twice the number of common lexical chains in the two documents.

The algorithm follows a complete link strategy in assigning a document to a cluster, *i.e.*, it assigns a document to a cluster only if the similarity of the document is above a threshold for all the documents in a cluster. The complete link strategy will result in clusters with smaller diameters [39] as compared to the single link approach which will result in elongated clusters. If a document is not added to any cluster, a new cluster is created and the document is added to it.

5.2 Evaluation

Evaluating clustering algorithms is always problematic. As elucidated in previous chapters and to the best of our knowledge, there does not exist a 'standardised' dataset for evaluating soft-clustering algorithms. For this reason, we resort to using a custom dataset derived from the 20 Newsgroups (20NG) dataset [9] for our evaluation. Since we are interested in a qualitative analysis of the clusters formed as opposed to a quantitative one, we keep the number of documents small in order to be able manually analyse them. To the best of our knowledge, no such qualitatively analysed benchmarks are available where the clustering obtained is assessed semantically with reference to topics discussed.

We selected 31 documents from *comp.graphics*, *talk.politics.guns*, *talk.mideast*, *talk.religion.misc* and *rec.auto* groups of the 20NG dataset. From each group, we first picked documents at random, and in the case of some of these documents we picked up a few more documents which are related directly and indirectly to them. This was done to obtain a controlled set of clustered documents. Thus we selected 4, 9, 9, 5, 4 documents from *comp.graphics*, *talk.politics.guns*, *talk.mideast*, *talk.religion.misc* and *rec.auto* groups respectively. We included an outlier document in this set as well.

The lexical chains were computed using the scheme described in Chapter 3. Consistent with our findings there, we have used only the lexical chains formed using nouns. We apply the

Cluster ID	Documents
Cluster #1	101557, 101597
Cluster #2	101574, 101597
Cluster #3	101677
Cluster #4	37261, 38400
Cluster #5	38406
Cluster #6	38460
Cluster #7	53294, 53354
Cluster #8	54152, 54455
Cluster #9	54206
Cluster #10	54253, 54269, 54358, 54819
Cluster #11	75414, 82783
Cluster #12	75933, 76184, 76506
Cluster #13	76099, 76486
Cluster #14	76227, 76506
Cluster #15	76289, 76306
Cluster #16	82781, 82782, 82783, 82784, 82785

Table 5.1: Soft-clustering results

chain selection rule described in Chapter 3 to prune the lexical chains. We ran the algorithm on this collection of 31 documents with a threshold value of 0.2 and the results obtained are shown in Table 5.1.

5.3 Discussion

We will now examine the performance of the algorithm by analysing the clusters formed. Both the documents in Cluster 4 are 'Call for Papers/Presentations' type documents and therefore are correctly grouped. Cluster 5 contains the outlier document. We now focus on clusters 11 and 16 which share a document. Cluster 16 contains a set of five documents which forms an email thread. One of these documents (82783) diverges to talk about 'conflict' which is the content of 75414.

While these results are interesting, it is far from perfect. Consider clusters 1, 2 and 3. While clusters 2 and 3 have the correct selection of documents, we are not so happy with cluster

1. Though both documents in cluster 1 talk about Toyota cars, it would have been better if either cluster 1 contained only 101557 or alternatively contained all three documents - 101557, 101597 and 101574.

The lexical chaining process can be considered to aggregate and summarise the information contained in the document. We feel that the idea of preprocessing the data to unravel the information contained in it, is a better strategy than forcing the algorithm to do the same. Infact our soft-clustering algorithm is considerably simple when compared to the other popular clustering algorithms.

Usually, the document will be placed in the cluster with which it has maximum proximity. This will result in only the most dominant topical strand of the document being used. As discussed in the previous sections, the algorithm is capable of placing documents in multiple clusters. This results in two advantages - (*i*) we get clusters with more fine grained topics and (*ii*) we are not required to decide on which cluster the document should be put in. Consider clusters 12 and 14. They share document 76506. Both the clusters are related to Jews and Judaism. While cluster 12 mostly focuses on philosophy and ideology, cluster 14 has docs related to a more social commentary such as families, ghettos, ancestry, *etc*. Thus, the document is interestingly positioned in both clusters. This is useful for many potential applications.

We point out two issues with this algorithm. Firstly, the outcome of the algorithm is dependant on the sequence in which the documents are processed. This is a general problem with most sequential clustering algorithms [49]. Another issue is with choosing the threshold value. The results presented in this chapter are based on a threshold value of 0.2 which was arrived at by manually examining the results obtained for various threshold values. Deriving a technique to identify a suitable threshold for a given dataset would be an interesting solution and we leave it as a task for the future.

Clustering is a subjective task and these results should be interpreted in that light. The

results in this chapter demonstrate an aspect of lexical chains based document representation. We feel that this is only a first step in this direction and many interesting research directions can be pursued from this initial step.

CHAPTER 6

Conclusions

This thesis introduces a new method to represent document features using the concept of lexical chains. We proposed a new algorithm to compute lexical chains. The key differences of our algorithm from previous algorithms is the use of an explicit WSD algorithm to disambiguate the words as a preprocessing step before the lexical chains are computed. This helped reduce the complexity of the lexical chaining algorithm considerably by eliminating many of the heuristics used in the previous algorithms.

We also introduced the notion of the global set which stored all relevant statistics along with the lexical chains identified for each document. This helped bring down the number of passes required over the input documents to one. This also made the lexical chaining algorithm incremental. We showed how feature vectors can be formed using these lexical chains. Namely, we proposed a novel technique for identifying 'good' chains to represent documents. A new feature weighting mechanism was also introduced.

We showed that representing documents using lexical chains improves the performance of clustering algorithms considerably. This is because these features are able to model the topics contained in documents better than the traditional bag of words representation. The previous literature on lexical chains implicitly assumed that using nouns alone will be sufficient to compute lexical chains. We have empirically shown that this is true.

We also investigated the use of other wordnet relations such as hypernyms and co-ordinate terms. We found that they do not add much to the expressive power of the lexical chains for the tasks that we consider in this thesis. Our chain selection strategy made these more complicated

lexical chain computing strategies redundant. Ignoring the other relations also resulted in a much easier and faster computation of the chains.

While the results on the clustering task has been very good, the numbers for the classification task is neutral. We attribute this to the inherent capability of SVMs to handle large dimensional feature vectors. Since SVMs have now become the de-facto standard for text classification, we do not feel that it is worthwhile to experiment with other classification algorithms. But, we do point out that the smaller dimensions resulting from the lexical chain based feature vectors results in much faster training for the SVMs. As discussed previously, the SVMs took many orders longer to work on the BoW features when compared to the lexical chains based features.

A direct competitor to the work presented in this thesis is the Latent Semantic Analysis technique. The technique is more mathematically oriented and has the advantage of being purely data driven. We have shown that the lexical chain based approach is able to beat the performance of the LSA approach. LSA requires a large volume of data to function effectively. This is not the case with the lexical chains based approach as we can compute lexical chains even with a small number of input documents. Extracting features by our method is also computationally much faster.

Documents contain multiple topics and clustering them using hard clustering is very unnatural. Documents should ideally be clustered using soft clustering algorithms. Unfortunately, these algorithms work only for very small dimensions. The nature of lexical chains makes them suitable for clustering documents. Each lexical chain is considered as a topical strand and if two documents share a chain, then they contain the same topics. We proposed an algorithm based on this idea for soft clustering documents. An evaluation was done on a small dataset and the results are very encouraging. One advantage of the algorithm is that we do not have to specify the number of clusters *a priori*. This is one of the most trickiest parameters of most conventional clustering algorithms.

Finally, the vector space formed by the lexical chain based features are more semantically oriented than just the Bag of Words features. This is advantageous for many applications such as topic detection and title generation.

6.1 Future Work

Our approach to representing document features using lexical chains can be described as a type of dimensionality reduction. We have not considered it from such an angle because dimensionality reduction is usually performed as a post-processing step to feature extraction while our approach does not work like that. Nevertheless, it would be interesting to compare how the lexical chains based features perform against the state-of-art feature selection mechanisms. This will be a significant effort in itself and is left for the future.

The soft clustering algorithm presented in this thesis is very promising. One possible direction of work on that would be to make it more invariable to the input sequence of the documents. A more elaborate mechanism to evaluate the similarity between documents and clusters might be helpful as well.

The use of lexical chains as a document representation mechanism holds promise for many applications. We list two among them:

Topic Detection There has been a large body of work previously done on Topic Detection and Tracking [50]. We feel that our work on lexical chains based document representation can be exploited for this purpose. At a very simple level, if we consider each (or a group of) chain(s) as a topical strand, then when a document is added to a chain, it will 'signal' the topic of the document. This is only a very nascent line of thought and stronger methods will be required to fine tune this approach.

Title Generation Each lexical chains represents a topic contained in the document. This fact could be used to generate titles for a single document or a group of documents by post-processing the set of lexical chains assigned to a document. If we can overlay the concept hierarchy from WordNet on this, we might be able to further generate a hierarchical set of titles.

References

[1] R. Xu and D. Wunsch II, "Survey of clustering algorithms," *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645– 678, May 2005.

[2] G. Salton, A. Wong, and C. S. Yang, "A vector space model for automatic indexing," *Commun. ACM*, vol. 18, no. 11, pp. 613–620, 1975.

[3] C. D. Manning and H. Schutze, *Foundations of Natural Language Processing*. MIT Press, 2002, ch. 15.

[4] M. F. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[5] M. A. K. Halliday and R. Hasan, *Cohesion in English*. Longman, 1976.

[6] J. Morris and G. Hirst, "Lexical cohesion computed by thesaural relations as an indicator of the structure of text," *Computational Linguistics*, vol. 17, no. 1, pp. 21–48, 1991.

[7] B. J. Grosz and C. L. Sidner, "Attention, intention and the structure of discourse," *Computational Linguistics*, vol. 12, no. 3, pp. 175–204, Jul-Sept 1986.

[8] C. Fellbaum, Ed., *WordNet: An Electronic Lexical Database*. MIT Press, 1998. [Online]. Available: wordnet.princeton.edu

[9] J. Rennie, "20 Newsgroups dataset," Online: http://people.csail.mit.edu/jrennie/20Newsgroups/, 1995.

[10] R. Barzilay and M. Elhadad, "Using lexical chains for text summarization," in *In Proceedings of the Intelligent Scalable Text Summarization Workshop (ISTS'97), ACL, Madrid, Spain.*, 1997. [Online]. Available: citeseer.ist.psu.edu/barzilay97using.html

[11] H. G. Silber and K. F. McCoy, "Efficiently computed lexical chains as an intermediate representation for automatic text summarization." *Computational Linguistics*, vol. 28, no. 4, pp. 487–496, 2002.

[12] G. Hirst and D. St-Onge, "Lexical chains as representation of context for the detection and correction of malapropisms," in *WordNet:An electronic lexical database and some of its applications.*, C. Fellbaum, Ed. The MIT Press, Cambrige, MA., 1997. [Online]. Available: citeseer.ist.psu.edu/hirst97lexical.html

[13] S. J. Green, "Automatically generating hypertext in newspaper articles by computing semantic relatedness." in *NeMLaP3/CoNLL98: New Methods in Language Processing and Computational Natural Language*, D. Powers, Ed., 1998.

[14] G. Ercan and I. Cicekli, "Using lexical chains for keyword extraction," *Inf. Process. Manage.*, vol. 43, no. 6, pp. 1705–1714, 2007.

[15] N. Stokes, "Applications of lexical cohesion analysis in the topic detection and tracking domain," Ph.D. dissertation, National University of Ireland, Dublin, 2004.

[16] M. Jarmasz and S. Szpakowicz, "Not as easy as it seems: Automating the construction of lexical chains using roget's thesaurus," in *Proceedings of the 16th Canadian Conference on Artificial Intelligence*, 2003.

[17] B. Kirkpatrick, *Roget's Thesaurus of English Words and Phrases*. Penguin, 1998.

[18] M. Jarmasz and S. Szpakowicz, "The design and implementation of an electronic lexical knowledge base," in *Proceedings of the 14th Biennial Conference of the Canadian Society for Computational Studies of Intelligence (AI 2001), Ottawa, Canada*, June 2001, pp. 325–334.

[19] "Cognitive Science Laborartory at Princeton University," Website: http://www.cogsci.princeton.edu/.

[20] "WordNets in the World," Online: http://www.globalwordnet.org/gwa/wordnet_table.htm.

[21] N. Ide and J. Veronis, "Word sense disambiguation: The state of the art," *Computational Linguistics*, vol. 24, no. 1, 1998.

[22] M. Lesk, "Automatic sense disambiguation using machine readable dictionaries," in *Proceedings of SIGDOC*, 1986.

[23] S. Banerjee and T. Pedersen, "Extended gloss overlaps as a measure of semantic relatedness," in *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, Acapulco, Mexico, August 2003, pp. 805–810.

[24] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison-Wesley, 2006, ch. Cluster Analysis: Basic Concepts and Algorithms.

[25] S. Sra, H. Cho, I. S. Dhillon, and Y. Guan, "Minimum sum-squared residue co-clustering of gene expression data." in *Proceedings of the Fourth SIAM International Conference on Data Mining, Lake Buena Vista, Florida, USA, April 22-24, 2004*, 2004.

[26] N. Cristianini and J. Shawe-Taylor, *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

[27] M. Steinbach, G. Karypis, and V. Kumar, "A comparison of document clustering techniques," in *KDD Workshop on Text Mining*, 2000. [Online]. Available: citeseer.ist.psu.edu/steinbach00comparison.html

[28] G. Karypis, "Cluto - a clustering toolkit," University of Minnesota - Computer Science and Engineering, Tech. Rep. 02-017, April 2002.

[29] I. S. Dhillon, S. Mallela, and D. S. Modha, "Information-theoretic co-clustering," in *Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining(KDD-2003)*, 2003, pp. 89–98. [Online]. Available: citeseer.ist.psu.edu/dhillon03informationtheoretic.html

[30] I. S. Dhillon, "Co-clustering documents and words using bipartite spectral graph partitioning," in *Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, San Fransico, California, USA, August 2001, pp. 26–29.

[31] H. Cho, I. Dhillon, Y. Guan, and S. Sra, "Minimum sum-squared residue co-clustering of gene expression data," in *Proceedings of The fourth SIAM International Conference on Data Mining*, April 2004, pp. 114–125.

[32] H. Cho, Y. Guan, and S. Sra, "Co-clustering software," Online: http://www.cs.utexas.edu/users/dml/Software/cocluster.html.

[33] D. D. Lewis, "Reuters 21578 text categorization test collection," Online at http://www.daviddlewis.com/resources/testcollections/reuters21578/.

[34] D. D. Lewis, Y. Yang, T. Rose, and F. Li, "RCV1: A new benchmark collection for text categorization research," *Journal of Machine Learning Research*, vol. 5, pp. 361–397, 2004.

[35] P. Pantel and D. Lin, "Document clustering with committees," in *SIGIR '02: Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. New York, NY, USA: ACM Press, 2002, pp. 199–206.

[36] B. Efron and R. Tibshirani, *An Introduction to the Bootstrap*. CRC Press, 1993.

[37] R. A. Johnson, *Probability and Statistics for Engineers*, 7th ed. Prentice-Hall India, 2005.

[38] G. Casella and R. L. Berger, *Statistical Inference*, 2nd ed. Duxbury Press, 2001.

[39] W. B. Frakes and R. Baeza-Yates, Eds., *Information retrieval: data structures and algorithms*. Upper Saddle River, NJ, USA: Prentice-Hall Inc., 1992.

[40] S. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman, "Indexing by latent semantic analysis," *Journal of the Society for Information Science*, vol. 41, no. 6, pp. 391–407, 1990.

[41] *Linear Algebra and its Applications*. Brooks Cole, 1988.

[42] V. Hatzivassiloglou, L. Gravano, and A. Maganti, "An investigation of linguistic features and clustering algorithms for topical document clustering," in *SIGIR 2000*, 2000, pp. 224–231. [Online]. Available: citeseer.ist.psu.edu/hatzivassiloglou00investigation.html

[43] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 121–167, 1998. [Online]. Available: citeseer.ist.psu.edu/burges98tutorial.html

[44] T. Joachims, "Text categorization with support vector machines: learning with many relevant features," in *Proceedings of ECML-98, 10th European Conference on Machine Learning*, C. Nedellec and C. Rouveirol, Eds., no. 1398. Chemnitz, DE: Springer Verlag, Heidelberg, DE, 1998, pp. 137–142. [Online]. Available: citeseer.ist.psu.edu/joachims97text.html

[45] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[46] C. Cortes and V. N. Vapnik, "Support vector networks," *Machine Learning*, vol. 20, pp. 273–297, Nov 1995.

[47] B. E. Boser, I. Guyon, and V. Vapnik, "A training algorithm for optimal margin classifers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. ACM Press, 1992, pp. 144–152.

[48] C. W. Hsu, C. C. Chang, and C. J. Lin, "A practical guide to support vector classification," Online:http://www.csie.ntu.edu.tw/ cjlin/papers/guide/guide.pdf.

[49] S. Theodoridis and K. Koutroumbas, *Pattern recognition*. Academic Press, 2006.

[50] "Topic detection and tracking," http://projects.ldc.upenn.edu/TDT/.

LIST OF PUBLICATIONS

Refereed International Conference Paper

- Jayarajan, D., Deodhare, D., and Ravindran, B., "Lexical Chains as Document Features", *in Proceedings of International Joint Conference on Natural Language Processing (IJCNLP) 2008, Hyderabad, India*.

- Jayarajan, D., Deodhare, D., Ravindran, B., and Sarkar, S., "Document Clustering using Lexical Chains", *at the Workshop on Text Mining & Link Analysis (TextLink) 2007, Hyderabad, India.*

GENERAL TEST COMMITTEE

CHAIRPERSON:

Prof. P. Sreenivasa Kumar
Professor
Department of Computer Science and Engineering
I.I.T. Madras, Chennai - 600 036

GUIDE:

Dr. B. Ravindran
Assistant Professor
Department of Computer Science and Engineering
I.I.T. Madras, Chennai - 600 036

CO-GUIDE:

Dr. Dipti Deodhare
Head, AINN Group
Center for Artificial Intelligence & Robotics
Bangalore - 560 093

MEMBERS:

Dr. C. Chandrasekhar
Associate Professor
Department of Computer Science and Engineering
I.I.T. Madras, Chennai - 600 036

Prof. G Srinivasan
Professor
Department of Management Studies
I.I.T. Madras, Chennai - 600 036

CURRICULUM VITAE

1. NAME : Dinakar Jayarajan

2. DATE OF BIRTH : 15^{th} June, 1979

3. PERMANENT ADDRESS : Govind Bhavan,
Tillery,
Quilon, Kerala.
PIN - 691 001
Email: dinakar.jayarajan@gmail.com
Phone: +91 474 2740701

4. EDUCATIONAL QUALIFICATIONS

Bachelor of Technology (B.Tech.)

Year of Completion	:	2001
Institution	:	TKM College of Engineering, Quilon Kerala
Specialization	:	Computer Science and Engineering

Master of Science (M.S. by Research)

Institution	:	Indian Institute of Technology Madras
Registration Date	:	03-08-2005
Date of Completion	:	03-06-2009