Using Compression For Source Based Classification Of Text

by

Nitin Thaper

Bachelor of Technology (Computer Science and Engineering),
Indian Institute of Technology, Delhi, India. (1996)

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

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY.

February 2001

Author ____________________
Department of Electrical Engineering and Computer Science
February 6, 2001

Certified by ____________________
Shafi Goldwasser
RSA Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by ____________________
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Using Compression For Source Based Classification Of Text

by
Nitin Thaper

Submitted to the
Department of Electrical Engineering and Computer Science
on February 6, 2001 in partial fulfillment of the
requirements for the degree of Master of Science.

Abstract

This thesis addresses the problem of source based text classification. In a nutshell, this problem involves classifying documents according to "where they came from" instead of the usual "what they contain". Viewed from a machine learning perspective, this can be looked upon as a learning problem and can be classified into two categories: supervised and unsupervised learning. In the former case, the classifier is presented with known examples of documents and their sources during the training phase. In the testing phase, the classifier is given a document whose source is unknown, and the goal of the classifier is to find the most likely one from the category of known sources. In the latter case, the classifier is just presented with samples of text, and its goal is to detect regularities in the data set. One such goal could be a clustering of the documents based on common authorship.

In order to perform these classification tasks, we intend to use compression as the underlying technique. Compression can be viewed as a predict-encode process where the prediction of upcoming tokens is done by adaptively building a model from the text seen so far. This source modelling feature of compression algorithms allows for classification by purely statistical means.

Thesis Supervisor: Shafi Goldwasser
Title: RSA Professor of Computer Science and Engineering

Acknowledgments

First of all, I would like to thank my advisor, Shafi Goldwasser, for patiently suggesting different problems over the course of the summer till I finally homed in on this one. Besides suggesting the seed idea for this thesis, she kept up my motivation level through the initial false starts, ensuring that I eventually made progress.

I would also like to acknowledge a fruitful discussion with Amit Sahai wherein he suggested a useful modification to our original strategy.

I would like to thank Professors Santosh Vempala and Piotr Indyk for helpful discussions on different aspects of clustering.

I am also grateful to Dr. William Teahan for providing pointers to prior work on this problem and allowing the use of his Text Mining Toolkit source code.

Many thanks are in order to the inhabitants of MIT's Lab for Computer Science; for providing a stimulating and perhaps more important, a friendly environment. In particular, credit is due to my present and past officemates – Dimitris Mitsouras, Prahladh Harsha and Alantha Newman. Thanks for being there!

Last but certainly not the least, I would like to thank my family for all the many big, little things that have made me what I am today (in particular, the *RTU* joke that made sure I finished up on schedule!).

NITIN THAPER
Cambridge, Massachusetts
February 6, 2001

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid growth of the Internet has led to an explosion of online text. A natural fallout of this phenomenon has been the problem of efficiently classifying this huge bank of information. Ideally, we would like to have a situation where any new piece of text appearing on the Internet gets classified automatically, based on its contents/language/origins etc. The problem of content based text categorization, in particular, has received a lot of attention from the machine learning and information retrieval communities. Typical approaches extract "features" from documents and use the feature vectors as input to a machine learning scheme that learns to classify the documents. The features are usually words and some sort of selection process is applied to extract the most relevant ones. This *vector space* model has been quite successful and forms the basis for most of the keyword based search engines in use today.

In this thesis, we investigate the related, but nonetheless different, problem of source based text classification. As the nomenclature suggests, the intention is to classify documents based on their "source" rather than their content. In other words, documents are categorized according to "where they came from" instead of "what they contain". The most obvious application is classification based on authorship. But the notion could very well be extended to the language of a document or its origins. A related problem is that of authorship authentication, where the objective is to ascertain the authorship of a given piece of text. Although it seems to run contrary to the anonymity theme of the Internet, this has uses, especially in e-commerce, where presently there is no way of authenticating testimonials that customers supposedly write about online merchants. Other motivating applications are detecting digital copyright violations and mirrored copies of similar documents on the Internet. The problem of copyright detection assumes added importance in the light of the publishing industry's increasing shift towards "electronic books".

Viewed from a machine learning perspective, this classification problem has two natural variants. The use of predefined categories implies a "supervised learning" approach to classification, where pre-classified documents – which effectively define the categories – are used as "training data" to build a model that can be used for classifying new documents that comprise the "test data". This contrasts with "unsupervised learning", where there is no training data and clusters of like documents are sought amongst the test data. We will investigate both variants of the problem in this thesis.

It would be desirable to have a technique that solves this problem without trying to "understand" the semantics of the underlying text. In other words, we would like to capture the "style" of a piece of text in purely statistical terms and use that for our classification/authentication purposes.

1.1 Text Modelling

Before proceeding with any analysis, one needs to create some kind of a model for the text in question. This model is just an abstract representation of the text that tries to capture the nature of the source of the text. It could range from something as simple as a single parameter (say average word length) to something as complex as probabilistic finite state machines or grammars. Once we have decided on a suitable model (and a method for its extraction), the problem of source identification reduces to a traditional nearest-neigbour classification problem, ie, we just "compare" the model of the text being tested with the models of pre-classified text and declare the "closest" match to be the most likely source for the text. But the devil lies in the details. It is unclear how one could compare anything but the simplest of models (which capture the text only in terms of numbers). And it is not surprising that the simplest models also tend to be the crudest. They do not capture enough nuances of the source, resulting in erroneous classification.

The issue therefore is to come up with a technique that harnesses the information inside a model in a way that permits accurate classification.

1.2 Compression for Classification

Text compression techniques have traditionally been distinguished as dictionary-based versus entropy-based (or statistical). The dictionary-based techniques [38, 39] build up a dictionary of phrases occuring in the text and achieve compression by replacing the subsequent occurences of those phrases by references in the dictionary. The entropy-based methods tend to follow a predict-encode paradigm. The predictor (or modeller) generates a probability distribution for the upcoming token (based on the input seen thus far) and the encoder uses this distribution to encode the actual token encountered. The underlying idea is to encode frequently occuring tokens with less bits and the rarely occuring ones with more. Some examples in this vein are Huffman encoding [20] and PPM (Prediction by Partial Matching) [10].

Viewing compression as a predict-encode operation immediately suggests the possibility of using compression as an engine for classification. The underlying idea is to compress the given piece of text using models of the different sources. The correct source of the text is most likely the one yielding the maximum degree of compression. This paradigm allows one to use any model of one's liking (as long as one has a compression method that is able to use that model). Furthermore, it is likely that the better the model, the better the classification obtained.

1.2.1 Unsupervised classification

The problem of unsupervised classification (or clustering) is important in its own right. It arises naturally in scenarios where the sources have not been identified beforehand. A motivating example of the problem comes from the ubiquitous domain of Internet search queries. A typical search query on the Internet yields a large collection of documents. It would be highly desirable to cluster the search results into a small number of categories, based on some criterion. One such criterion could be common authorship of the documents, in which case, compression can be used in a natural way for determining the closeness between pairs of documents. Once the closeness metric has been computed, one can use any of the standard clustering algorithms to carry out the actual categorization.

1.3 Related Work

Content based text categorization has been investigated by a number of people using different techniques. Dumais et al [13] have used the Bayes net technique for this purpose. A decision tree approach using the divide and conquer strategy has been described in [27]. Ng et al have used neural nets for classification in [30]. Frank et al [17] used compression for this task but concluded that compression was not the best choice for this application because content is usually determined by a few important keywords and compression techniques tend to be insensitive towards particular words/phrases. The best possible technique seems to be the one using *linear support vector machines* [24]. Interestingly, apart from their practical utility, SVMs happen to be well founded in computational learning theory and are very open to theoretical understanding and analysis.

Clustering algorithms for clustering results of search queries have been described in [12],[25]. One such prototype of a clustering system has been implemented in [1], which uses common content as the clustering criterion. Under this criterion, two documents are deemed close if they have many common keywords.

Statistical techniques for carrying out authorship attribution have been in vogue in the social sciences for some time now, although there is no clear consensus on the best possible method. One of the common techniques is the "QSUM[1] method" [15]. This method works by successively adding up the deviations of sentence length and "language habits" (traits supposedly invariant for an author such as use of two and three letter words, and words beginning with a vowel) in a given sample and plotting them on a graph. The resulting graph is treated as a "fingerprint" for the author's writing style. This technique has been used in a number of court cases and has received significant public attention. However, a number of independent investigations have found the technique unreliable [19]. AI techniques such as neural nets have also been cited as possible candidates for this problem [37]. A compression based method, similar to ours, has been described in [36]. This paper describes the use of PPM based compression techniques for resolving the authorship attribution problem for two authors as well as for distinguishing different languages.

Copy detection mechanisms have emerged as important tools to prevent digital copyright violation on the Internet. Methods based on *watermarks* have been investigated in [9] and [6]. Shivakumar et al [32] propose a detection mechanism based on hashing "chunks" (the chunks could range from words to sentences) of documents and counting common chunks as a means to quantify overlap between documents.

1.4 Organization of the Thesis

We present a brief introduction to compression techniques in Chapter 2 and show their relevance to our problem. We also present our experimental results for the supervised classification problem and the related problem of copy detection. In Chapter 3, we give a brief background on the general problem of clustering and

[1]an abbreviation for cumulative sum

present our approach of applying clustering to the unsupervised classification problem. We also discuss our experimental results for this problem. Finally, in Chapter 4, we present our conclusions about the usefulness or otherwise of this idea.

Chapter 2

Compression for classification

Compression being our underlying technique for text classification, we now present a brief overview of the information theoretic basis of compression and it's relationship to our problem. The reader already comfortable with the basics of information theory may wish to skip to Section 2.3, wherein we describe our contribution of using compression as a means for classification. We discuss the internal mechanism of two commonly used compression techniques in Section 2.4 and present our experimental results in Section 2.6.

2.1 Entropy as a Measure of Information Content

The notion of *entropy* is a pivotal one in information theory. The entropy is a measure of the uncertainty in a message. It can also be considered a measure of the "information content" of the message – more probable messages convey less information than less probable ones. Formally speaking, the information content (or entropy), I of a message m with probability $p(m)$ is,

$$I(m) = log(\frac{1}{p(m)}) = -log\, p(m) \tag{2.1}$$

The information content per symbol is just $I(m)/|m|$.

In general, the probability distribution $p(m)$ of the messages generated by a source is hard to estimate. This is circumvented by using a model M as an approximation to the actual source. Let $p_M(m)$ be the probability of message m under the model M. The quantity $I_M(m) = log\frac{1}{p_M(m)}$ is called the *cross-entropy*[1]; the closer it is to $I(m)$, the better the modelling of the source by M.

[1]Sometimes just the fraction $\frac{1}{p_M(m)}$ is used in which case it's called the *perplexity* [8]

2.2 Entropy and Compression

The notion of entropy is easily generalized from messages to sources (or the languages generated by sources). For a source[2] generating messages from a finite message space, S, its entropy is just the expectation of the individual message entropies,

$$H(P) = -\sum_{s \in S} p(s) log\, p(s) \tag{2.2}$$

where $p(s)$ is the probability of message s and all the probabilities in the distribution P are assumed independent and summing up to 1.

The entropy of an arbitrary source can be defined in a similar fashion. Let S^n denote the set of all strings of length n from an alphabet S. Then the n^{th} *order normalized entropy* is defined as [3]:

$$H_n(P) = -\frac{1}{n} \sum_{s \in S^n} p(s) log\, p(s) \tag{2.3}$$

The source entropy is then defined as

$$H(P) = \lim_{n \to \infty} H_n(P) \tag{2.4}$$

In general, it is hard to determine the source entropy of an arbitrary source just by looking at the output, since it requires looking at extremely long output sequences. As mentioned earlier, we use a model to estimate this entropy, that is we actually compute $H_M(P)$.

It turns out that the entropy of a source is a lower bound on the average number of bits per symbol needed to encode a message generated by the source (the fundamental source coding theorem [31]). This ties together the notions of entropy and compression, by setting a target on the best possible compression physically realizable. The compression ratio[4] actually achieved, provides us with an estimate of the message cross-entropy. In particular, *by comparing the compression results derived from two competing models for a source*, it is straightforward to infer which model captures the "style" of the source better. This key insight provides us with an easy way of quantitatively evaluating the closeness of a piece of text to a given source – it is just the compression ratio of the text using a suitable model of the source.

[2]Such a source is technically called a *zeroth-order* or memoryless source.

[3]This is normalized through division by n so that it represents per symbol entropy.

[4]The compression ratio is the ratio of compressed message size to original message size and is usually expressed in bits per character (bpc)

2.3 Supervised Classification using Compression

Compression algorithms are either adaptive or static, depending on whether or not they change their behavior according to the input seen so far. The key requirement for this problem is a compression algorithm operable in both adaptive and static modes. During the model construction phase, such an algorithm should work adaptively and output its internal model at the end. During the cross-entropy computation phase, it should initialize itself with the previously created model, and use it to compress the text in question. The compression ratio achieved is an indicator of the "distance" between the text and the source. Once the cross-entropies with respect to all the candidate sources have been evaluated, a simple minimum distance classifier can be used to identify the right source. The overall classification methodology is summarized below.

```
SUPERVISED CLASSIFICATION
{
  Training Phase
  FOR each of the candidate sources, $Source_j$ DO
    run compression adaptively on sample texts of $Source_j$
    save the compression model, $M_j$

  Testing Phase
  Input:  document $D$
  Output:  likely source of the document

  FOR each of the candidate sources, $Source_j$ DO
    run compression statically on $D$ using training model $M_j$
    $I_j(D)$ = get compression ratio (cross-entropy)
  return arg min $I_j(D)$
}
```

The classification schema outlined above highlights the fact that this methodology is not limited to any particular compression technique. It provides the flexibility of choosing the compression method best suited to the domain in question.

2.4 Compression Techniques

In this section, we give a brief overview of some of the common compression techniques. As mentioned earlier, compression techniques have traditionally been classified as dictionary-based ones and entropy-based ones. In Section 2.4.1, we describe a couple of variants of the basic Lempel-Ziv dictionary based compression scheme. In Section 2.4.2, we describe the PPM algorithm, a classic example of an entropy-based compressor.

2.4.1 Lempel-Ziv Algorithms

The Lempel-Ziv algorithms compress by building a dictionary of previously seen strings. The dictionary is then used to encode these strings. The algorithms, which can be shown to be perfect in the information-theoretic sense, work as follows. Given a position in a file, search through the preceding part of the file to find the longest match to the string starting at the current position, and output some code that refers to that match. Move the cursor past the match and repeat. The two main variants of the algorithm were described by Ziv and Lempel in two separate papers in 1977 and 1978 [38, 39], and are usually referred to as LZ77 and LZ78. The algorithms differ in how far back they search and how they find matches.

The LZ77 algorithm is based on the idea of a sliding window. The algorithm only looks for matches in a window a fixed distance back from the current position. Some commonly used algorithms that use this technique are `Gzip`, `ZIP` and `V.42bis` (a standard modem protocol). However, the fact that this algorithm is necessarily adaptive, ie, it uses preceding information from the same document, makes it undesirable for our application.

The other variant, LZ78, used in the Unix `compress` utility, meets our requirements. A top-level description of the algorithm is given on the next page [29].

The algorithm starts with a dictionary containing one entry for each character. As the algorithm progresses, it adds new strings to the dictionary such that each string is only added if a prefix one character shorter is already in the dictionary. For example, `John` is only added if `Joh` had previously appeared in the text sequence. Furthermore, the prefix string in the text (`Joh` in this case) gets replaced by its index in the dictionary. This method allows the decoder to reverse the encoding steps by building up the same dictionary as the encoder and thus obviates the need for sending the entire dictionary along with the compressed text. This fact, although important in practical compression scenarios, is of little consequence to us as we are interested only in the encoding process.

```
LZ78 COMPRESS                          LZ78 DECOMPRESS
{                                      {
  str = get input character              read old_code
  WHILE more input DO                    output old_code
    c = get input character              WHILE more input DO
    IF str+c is in dictionary              read new_code
      str = str+c                          str = translation of new_code
    ELSE                                   output str
      output the code for str              c = first character in str
      add str+c to dictionary              add old_code+c to dictionary
      str = c                              old_code = new_code
  output the code for str
}                                      }
```

Usage for classification

The dictionary built up by the encoder during the training phase, serves as the language model for the testing phase. During the testing phase, cross-entropy calculations against this model are done by running the algorithm in a static mode. The algorithm is initialized with the training dictionary and the line of code highlighted above, is suppressed to prevent any modifications to the dictionary. The compression ratio achieved at the end of the compression run, gives us an estimate of the desired cross-entropy.

2.4.2 Prediction by Partial Matching

PPM has set the performance standard in lossless compression of text throughout the past decade. The original algorithm was first published in 1984 by Cleary and Witten [10], and a series of improvements were described by Moffat, culminating in a careful implementation, called PPMC, which has become the benchmark version [28].

The underlying idea of PPM is to generate a conditional probability for the current character by keeping track of the last k characters. In other words, a k^{th} order Markov model is used for the prediction of upcoming characters[5]. The simplest way to build such a model is to keep a dictionary for every possible string s of k characters, and for each string store counts for each character x that follows s. The

[5]These are also known as *k-gram* models in the literature

conditional probability of x in context s, $P(x|s)$ is then estimated by $C(x|s)/C(s)$, where $C(x|s)$ is the number of times x follows s and $C(s)$ is the number of times s appears. The probability distributions are then used by an Arithmetic Coder to generate a bit sequence (Arithmetic coding [5] is a generalization of Huffman encoding that encodes symbols optimally even when the symbol probabilities are not powers of $\frac{1}{2}$).

The actual implementation actually maintains contexts of all lengths upto k, and effectively combines the different distributions into one via the mechanism of *escape events*. When faced with a string that is absent in the dictionary of k-sized contexts, the algorithm tries to match a string of one shorter length. This switching to a lower-order model is known as an *escape event*. This process is repeated for shorter and shorter lengths until a match is found (hence the nomenclature PPM). In practice, a single *trie* data structure is used to keep track of the probabilities in the different context models. Table 2.1 shows the data structure generated on a sample piece of text.

Table 2.1: An order 2 PPM model after processing the string *accbaccacba*

Order k = 2			Order k = 1			Order k = 0		
Context	Next	Count	Context	Next	Count	Context	Next	Count
ac	b	1	a	c	3	-	a	4
	c	2					b	2
			b	a	2		c	5
ba	c	1						
			c	a	1			
ca	a	1		b	2			
				c	2			
cb	a	2						
cc	a	1						
	b	1						

It would seem that the higher the value of k, the better the achievable compression. However, the amount of storage required by PPM grows exponentially with k. This places a practical limit on how large k can be. Furthermore, the larger the size of a given context, the less frequently it occurs. Consequently, the probabilities derived from the frequency counts are not reliable enough (the *sparse data* problem). In practice, it has been observed that order 5 models tend to give the best compression results [34].

The basic idea of PPM can be extended to word based models where words are treated as individual symbols and the conditional probabilities are calculated in a similar fashion. The reason for considering these models is the intuition that writing style is captured better at the sentence level rather than at the word level.

Usage for classification

In our implementation, we modify the basic PPM algorithm slightly. Firstly, we skip the aritmetic coding step since the actual compressed output is not needed. During the training phase, the program populates the trie data structure with appropriate character frequencies and stores it for subsequent use. This data structure captures the style of the source in terms of probabilities and serves as the training model.

The mechanism for the testing phase is similar to that of the LZ78 scheme. The training model is loaded and the probability (or equivalently the entropy) of each token in the test document is calculated from the finite context probabilities in the training model.

It might be worthwhile to add here that the distinction between dictionary-based and entropy-based algorithms is somewhat contrived. The dictionary-based schemes can also be looked upon as following a model-encode approach where the two phases are somewhat merged together. The modeller generates an implicit probability distribution for upcoming tokens based on phrases in the dictionary and the replacement of a phrase by its reference corresponds to the encode operation.

2.5 Utility of Compression Models

At this stage, it might be worthwhile to examine whether it is really meaningful to model the kinds of "sources" we are interested in by dictionaries or finite order Markov models. Firstly, the probability, $p_M(s)$, of a piece of text, $s_1 s_2 \dots s_n$, calculated using this approach is nothing but,

$$p_M(s) = \prod_{i=1}^{n} p_M(s_i | s_{i-k} s_{i-k+1} \dots s_{i-1}) \tag{2.5}$$

This expression brings out one underlying assumption, namely, the *locality* in human generated text. This is usually attributed to the attention span of the human mind while writing a piece of text. What comes next is most likely affected by what was written immediately, not far, before. However, sometimes the grammatical constructions of a particular language force "long range" effects. For example, the form of the verb at the end of a sentence can be affected by the kind of subject at the beginning – in English, by the plurality of the subject, in Hindi, by the subject's gender. Such effects will be missed by a finite context model. An appealing alternative is grammar-based langauge models. Grammar has long been the representation of language used in linguistics and intuitively such models capture properties of language that n-gram models cannot. Researchers have proposed different methods for inducing probababilistic grammars[6] from text [8, 26]. However, the current state-of-the-art methods either do not outperform $n-$gram models or do so marginally, with a high computational overhead.

The other issue is specific to the authorship attribution problem. The implicit assumption here is that humans have a subconsciously determined writing style, which is relatively invariant. Debating the existence or otherwise of subconscious writing cues is beyond the scope of this work – we choose to accept this as a worthwhile assumption from common day-to-day experience. The invariance of the writing style is somewhat harder to defend. Indeed, an author's writing style might differ markedly over a period of time or over different genres of writing. While this definitely limits the applicability of our technique, on the plus side, this deviation might actually offer a way of dating an author's different works or classifying them into different genres. Indeed, one of the objectives of this thesis is to experimentally investigate the validity of these assumptions.

2.6 Experimental Results

In this section, we describe our experimental setup and results. Subsection 2.6.1 describes our experiments in the domain of supervised source classification. This includes the problems of authorship and language identification. Subsection 2.6.2 discusses the copy detection problem.

2.6.1 Supervised Classification

Authorship attribution

For this experiment we started with a corpus of relatively large-sized documents written by ten authors (see Table B.1 in Appendix B for the list of authors and their works). The corpus was partitioned into training and testing sets and the training

[6]Grammars with probabilities attached to each of the rules

set was used to generate the models. Subsequent compression runs compressed each document in the testing set with each of the models and the author corresponding to the model with the best compression was ascribed to the document (since we had a limited number of works for each author, we repeated this process for different possible partitions of the data set and accumulated the results of each run into one aggregate result).

We tried out different compression techniques for this experiment. Method A uses a version of the LZ78 dictionary based algorithm. Method B uses code from the Text Mining Toolkit [35] and is technically an order 5 character based PPM technique. Method C makes use of code from the CMU-Cambridge Statistical Language Modelling Toolkit [2] and is essentially an order 3 word based PPM technique.

The accuracy for each method was determined by counting the number of correctly classified documents. These are summarized as percentages in Table 2.2

Table 2.2: Identification accuracy for different compression methods

Method A	Method B	Method C
73.3%	**85.0%**	35.0%

The character based PPM technique turns out to be the best performing method with an accuracy of 85%. When compared with an accuracy rate of 10% for random guessing , it seems to indicate that this method can extract "style" to a high degree. The low accuracy rate of the word based PPM technique can be attributed to the sparsity of training data, since this method treats each word as a symbol.

Table 2.3 illustrates the typical cross-entropies obtained for a sample testcase. Both the actual author of the testcase and the minimum cross-entropy number are highlighted. Note that the work is correctly ascribed to its author in this case.

Table 2.3: Cross entropies for the document *Hamlet* (in bpc)

Austen	Defoe	Dickens	Emerson	Kipling	**S'peare**	Shaw	Twain	Wells	Wilde
3.263	3.236	3.203	3.237	3.192	**2.659**	3.132	3.161	3.236	3.060

Table 2.4 illustrates an example of a misclassified work.

It is worthwhile to analyse this example in some detail. In this case, the training set for Kipling comprised of the following works: *Kim, Tales from the Hills* and

Table 2.4: Cross entropies for the document *Captains Courageous* (in bpc)

Austen	Defoe	Dickens	Emerson	**Kipling**	S'peare	Shaw	Twain	Wells	Wilde
3.144	3.090	2.869	3.061	2.820	3.338	2.868	2.823	**2.789**	2.951

American Notes. Both *Kim* and *Tales from the Hills* have colonial India as a backdrop and the writing contains many idioms and stylistic elements of Indian origin. On the contrary, *Captains Courageous* is a seafaring adventure drama with an American backdrop. Therefore it is not surprising that the cross-entropy numbers for this work indicate a closer match with H. G. Wells and Mark Twain, the other exponents of the adventure genre in this collection of authors.

It is also instructive to examine the misclassified documents to see how "close" they are to being classified correctly. Define a testcase to be of rank i if the testcase's author ranks i^{th} in its cross-entropy vector. In terms of ranks, the overall accuracy rate is nothing but the percentage of rank 1 documents. A histogram displaying the relative percentages of different ranks is shown in Figure 2-1. The histogram shows that most of the misclassified documents actually had rank 2, implying that the likelihood of the real author being among the first few minimum values is very high.

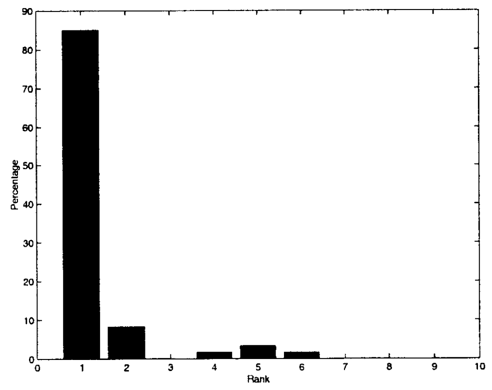

Figure 2-1: Histogram for different ranks

Dependence on document size

In order to observe the relationship between identification accuracy and document size, we repeated the experiment for different training and testing set sizes. Figure 2-2 shows the variation in accuracy as the training and testing set sizes are varied.

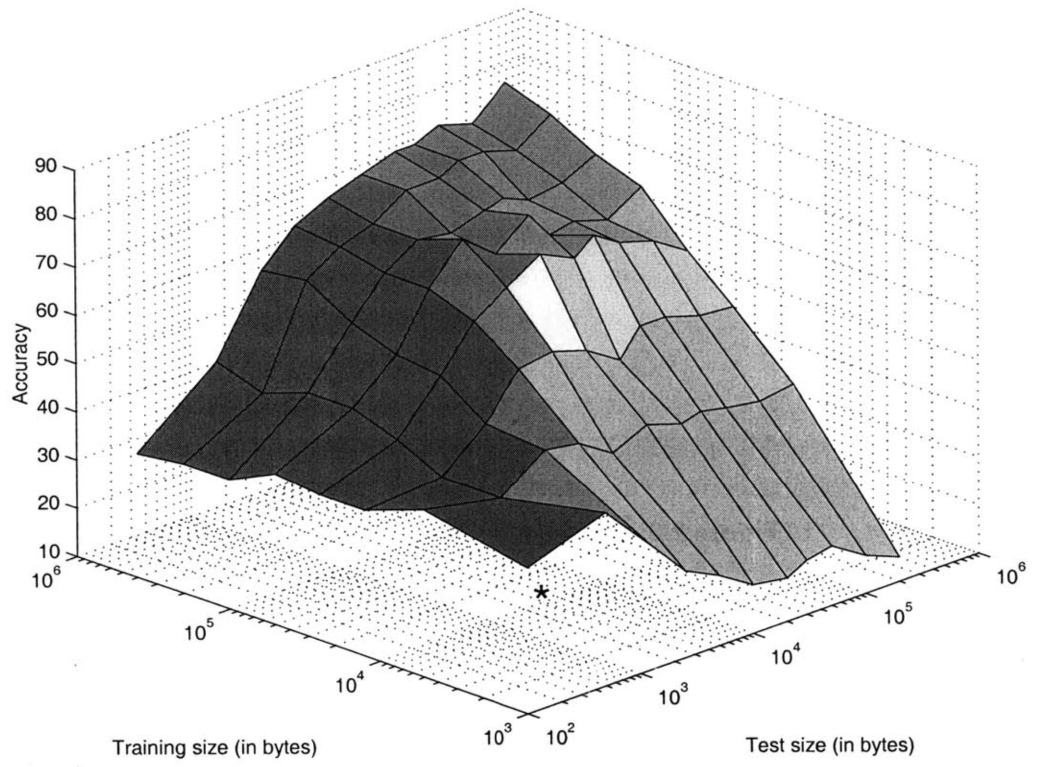

Figure 2-2: Variation of accuracy with training and testing set sizes for method B (The document sizes are plotted on a log scale)

Not surprisingly, the accuracy drops with decrease in training and/or testing set size. Figures 2-3 and 2-4 show the variation in accuracy with training (testing) set size for a fixed testing (training) set size. A distinctive feature of these graphs is the presence of a "threshold" phenomenon. The accuracy remains relatively stable for a wide range of training (testing) sizes and then falls sharply when the document size drops below a threshold value. This phenomenon allows for robust classification over a wide range of testcase sizes, as long as they are above the threshold value.

Incidentally, there is an interesting anomaly in the data (indicated by a $*$ in Figure 2-2), which is worth pointing out. Although, accuracy does drop off with decreasing document sizes, for very low values of training (testing) set sizes, one observes an increasing accuracy for decreasing testing (training) sizes (Figure 2-5). This seems to indicate that for small document sizes, the best performance is achieved when the training and testing set sizes are "matched". We have no

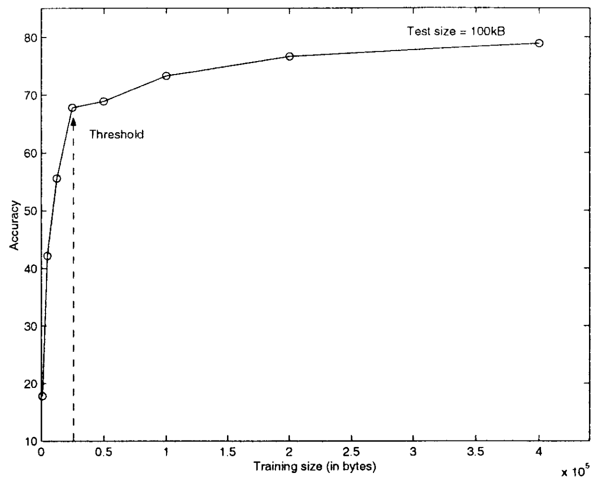

Figure 2-3: Accuracy vs Training set size

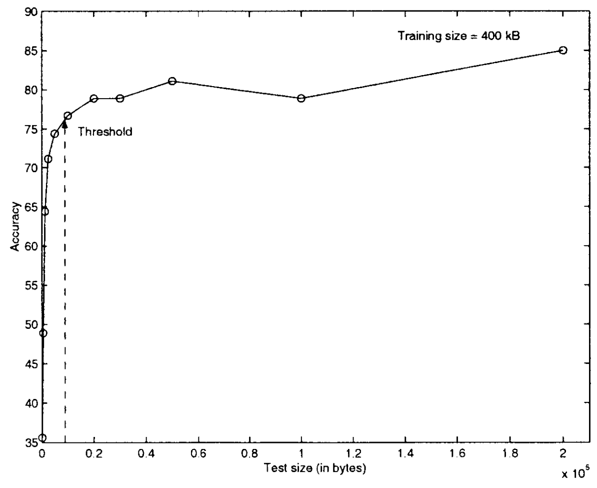

Figure 2-4: Accuracy vs Testcase size

explanation at this point for why this should be the case.

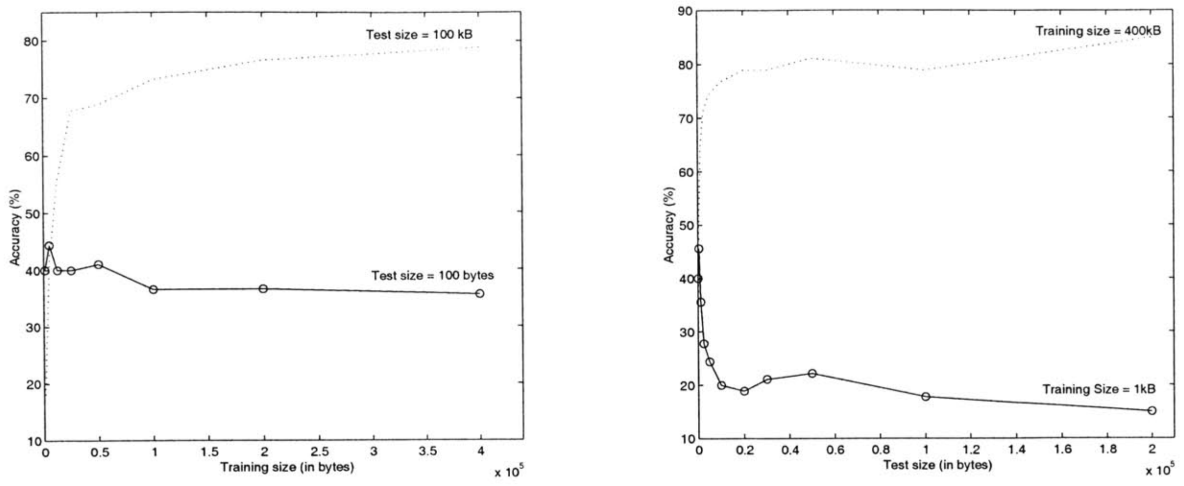

Figure 2-5: Anomalous behavior for small Training/Test sizes

We performed a similar experiment in the somewhat restricted domain of technical writing. In particular, we looked at technical papers and surveys written by ten researchers in the field of theoretical computer science. Since most of the documents were in Postscript format, they were converted into plain text by a preprocessing utility. Admittedly, this does result in the loss of certain information, especially regarding figures and special symbols, but we feel this is not too significant. Table 2.5 shows the identification accuracies for this experiment.

Table 2.5: Identification accuracy in the case of technical documents

Method A	Method B	Method C
62.5%	**75.0%**	40%

The slightly lower accuracy results can be attributed to the limited size of training documents and the fact that many documents had overlapping content because of the restricted domain of writing.

Language Identification

One would expect the language identification problem to be easier compared to the author identification one, since the grammar and vocabulary of different languages, induce radically different distributions on the space of possible sentential forms. This intuition is indeed borne out in our experiments.

We used text from ten different languages: Dutch, English, French, German, Italian, Latin, Portugese, Sanskrit, Spanish and Tamil (For the Indic languages, Sanskrit and Tamil, we used their transliterated versions in Roman script). The accuracy rate, not surprisingly, turned out to be 100% for both methods A and B, while it was 60% for method C.

2.6.2 Copy Detection

As mentioned in Chapter 1, there is a growing need to detect copies of digital content on the Internet, both from the standpoint of prevention of copyright violations as well as pruning document searches by weeding out very similar documents. We propose the following framework for solving this problem. Given a document d, we define it's *self-entropy* as the entropy calculated by compressing it with its own compression model. We expect that the cross-entropies of documents with "minor" deviations are close to the self-entropy, while those of unrelated documents are not. The degree of closeness can be specified as a percentage of the self-entropy.

We looked at one example in this vein. We performed a search on "Perl manuals" and focussed on three particular results, Perl version 5 manual, Perl version 4 manual and the Perl FAQ. Perl 5 being a major revision of Perl 4, the manuals are different from each other but still retain similar content in most parts. Treating the version 5 manual as our reference copy we calculated the self and cross entropy numbers (we used method A for this as it is well suited for exploiting sequences of common phrases). Table 2.6 summarizes the results.

Table 2.6: Entropy numbers for the Perl example

Document	Entropy
Perl v5 Manual	**4.16**
Perl v4 Manual	4.57
Perl FAQ	6.16
Unrelated document	6.22

Thus, a 10% cutoff for determining copies would be sufficient to classify the version 4 manual as a copy in this case. In general, the choice of the cutoff would be determined by the actual self-entropy number and its gap with the cross-entropy of unrelated documents. This idea could also be used for identifying related threads in a discussion forum such as a newsgroups; since newer threads tend to contain quoted text from older ones and hence should have similar entropy numbers.

2.6.3 The Shakespeare Authorship Controversy

As a diversion, we looked at the controversy surrounding Shakespeare's works. There is considerable dispute in certain literary circles regarding the identity of the "real" author of Shakespeare's works. Of the more than eighty Elizabethans put forward since the eighteenth century as the "true Shakespeare", the ones that have merited serious consideration are Sir Francis Bacon, Chritopher Marlowe, William Stanley (6th Earl of Derby), and Edward de Vere (17th Earl of Oxford). We focussed on Bacon as a possible contender, since his work was available digitally. We computed the cross-entropies between Shakespeare's own works and between Shakespeare's and Bacon's works, the intuition being that if Bacon was indeed the real author of works attributed to Shakespeare, we would get similar numbers. The results are shown in Table 2.7.

Table 2.7: Cross entropies of Shakespeare's works with two different training sets

Shakespeare's training set	Bacon's training set
2.680	3.195

The gap between these entropy numbers provides evidence, although by no means conclusive, that Bacon was not the author of these works.

Chapter 3

Clustering

Classification of documents based on common sourcehood, without any *a priori* identification of the sources, is a problem of clustering. It arises naturally when documents of unknown origin need to be segregated on the basis of their language or authorship. A mechanism for doing such segregation automatically would be highly desirable.

The problem of clustering is not unique to our domain; it arises in many areas [22]. The general clustering problem asks for clusters of "like" data points, where the likeness between data points is captured via a suitable distance metric. Although the problem itself is easy to state and understand conceptually, its mathematical formulation is not so straightforward. One debatable issue is the quality measure that should be optimized. We discuss this issue in Section 3.1. Numerous algorithms have been proposed by researchers for the clustering problem. We present our choice and describe it in detail in Section 3.2. In Section 3.3, we present our contribution to this problem, wherein we introduce the idea of using compression to determine the likeness between documents. Finally, in Section 3.4, we describe the results of our experiments.

3.1 Optimization Criteria for Clustering

Researchers have investigated different combinatorial measures of clustering quality. These include *minimum diameter* (minimize the largest distance between any two points in a given cluster), *minimum sum* (minimize the sum of distances between points in the same cluster), *k-center* (minimize the maximum distance from a point to its nearest cluster center), *k-median* (minimize the sum of distances from each point to its nearest cluster center) [7, 14, 21, 23]. There is no *a priori* reason to select one metric over the other, the usefulness of a given metric is predicated on the particular problem at hand. However, Kannan et al [25] argue that each of the metrics cited above are easy to fool (Figures 3-1 and 3-2). They propose a new bicriteria measure of the quality of a clustering which we describe below.

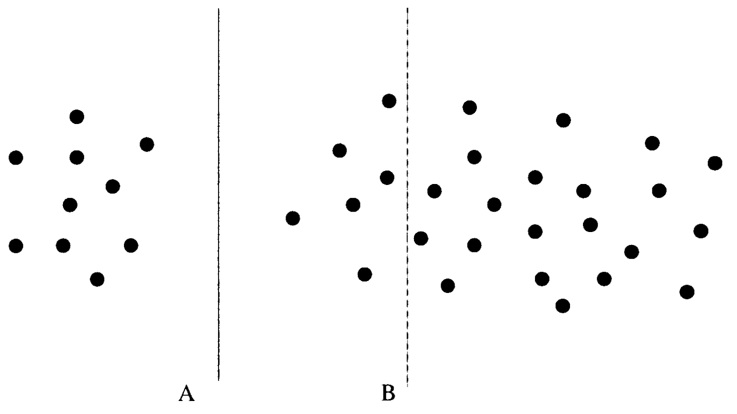

Figure 3-1: Minimizing the diameter produces the clustering B while A is clearly more desirable.

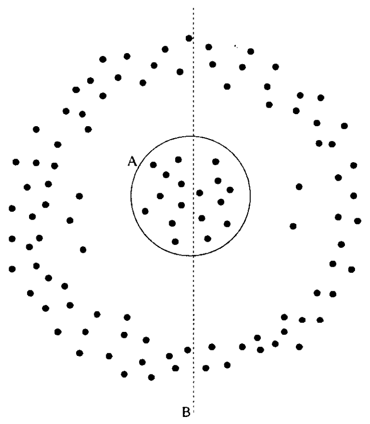

Figure 3-2: The inferior clustering B is produced by optimizing the minimum sum, 2-center or 2-median measures while A again the more suitable one.

3.1.1 Conductance as a Clustering Measure

The clustering problem is mathematically modelled via an edge-weighted complete graph whose vertices need to be partitioned. The weight of an edge a_{ij} represents the similarity of the vertices i and j; the higher the weight, the greater the similarity.

A cluster is just a subgraph of the similarity graph. The quality of a cluster should be determined by how similar the points within a cluster are. In particular, if there is a cut of small weight that divides the cluster into two pieces of comparable size then we expect the cluster to have low quality. This suggests the choice of minimum cut as a clustering metric. However, just the size of the minimum cut alone is misleading as illustrated in Figure 3-3.

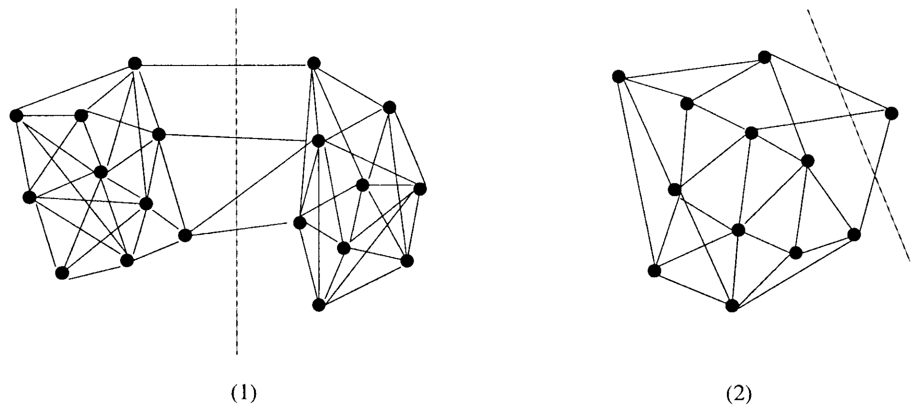

Figure 3-3: The first subgraph is a lower quality cluster than the second one even though it has a higher mincut value.

We would like to find a cut whose weight is small *relative to the sizes of the pieces it creates*. A quantity that measures the relative cut size is the *expansion*. The expansion of a graph is the minimum ratio over all cuts of the graph of the total weight of edges of the cut to the number of vertices in the smaller part created by the cut. Formally, we denote the expansion of a cut $(S, \bar{S})$ by:

$$\psi(S) = \frac{\sum_{i \in S, j \notin S} a_{ij}}{min(|S|, |\bar{S}|)} \tag{3.1}$$

The expansion of a graph is defined to be the minimum expansion over all the cuts of the graph. As a first step, we can define the quality of a cluster as the expansion of the subgraph corresponding to it. The expansion of a clustering is the minimum expansion of one of the clusters.

However, the measure defined above gives equal importance to all the vertices of the given graph. This may skew the obtained clusters in an undesirable way. For example, even if vertex i has very little similarity to all the other vertices in its cluster (ie, $\sum_j a_{ij}$ is small), it will still remain a part of that cluster because the cut that separates it has high expansion (Figure 3-4).

Therefore, it is prudent to give greater importance to vertices that have many similar neighbors and lesser importance to ones with few similar neighbors. This

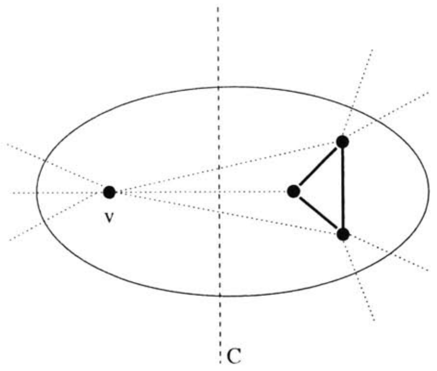

Figure 3-4: Cut C that separates vertex v from the rest of the cluster has high expansion because the size of the smaller subset is just 1.

can be done by a direct generalization of the expansion, called the *conductance*, in which subsets of vertices are weighted to reflect their importance.

The conductance of a cut $(S, \bar{S})$ in $G = (V, E)$ is denoted by:

$$\phi(S) = \frac{\sum_{i \in S, j \notin S} a_{ij}}{min(a(S), a(\bar{S}))} \tag{3.2}$$

where $a(S) = a(S, V) = \sum_{i \in S} \sum_{j \in V} a_{ij}$. The conductance of a graph is the minimum conductance over all the cuts of the graph;

$$\phi(G) = \min_{S \subseteq V} \phi(S) \tag{3.3}$$

The definition of conductance needs to be generalized further in order to quantify the quality of a clustering. For a given cluster $C \subseteq V$ and a cut $(S, C \backslash S)$ within C, where $S \subseteq C$, the conductance of S in C is defined analogously as:

$$\phi(S, C) = \frac{\sum_{i \in S, j \notin C \backslash S} a_{ij}}{min(a(S), a(C \backslash S))} \tag{3.4}$$

The conductance of a cluster $\phi(C)$ is then the smallest conductance of cut within the cluster. The conductance of a clustering is the *minimum* conductance of its clusters. This leads to the following optimization problem: given a graph and an integer k, find a k-clustering of *maximum* conductance.

3.2 Clustering Algorithms

Clustering algorithms are usually classified as *bottom-up* (also called agglomerative) or *top-down* (also called divisive). As the names suggest, the difference lies in whether the clusters are built *up* from individual data points or partitioned *down* from the initial single collection. The clustering algorithm terminates when it has found a clustering with the desired number of clusters. However, in clustering problems like ours, the number of clusters is not known *a priori*. A bad choice of k, the number of desired clusters, can lead to a poor clustering. In particular, if we use conductance as a measure of clustering quality, and ask for a k-clustering, where k is less than the "actual" number of clusters, we will end up with low conductance clusters.

One way to handle this problem is to avoid restricting the number of clusters. But then the algorithm can simply return each data point as a separate cluster ! This observation motivates the measurement of the quality of a clustering using two criteria [25]. The first is the minimum conductance of the clusters (called α), and the second is the fraction of the total weight of edges that are not covered by the clusters (called ϵ). Intuitively, we'd like to keep the conductance high while simultaneously minimizing the weight of edges not present in the clusters. This will prevent the generation of unnecessary clusters. Formally, let us call a clustering $(C_1, C_2, \dots, C_l)$ of V an (α, ϵ)-clustering if:

- The conductance of each C_i is at least α.
- The total weight of inter-cluster edges is at most an ϵ fraction of the total edge weight.

This leads us to the following bicriteria optimization problem:
Find an (α, ϵ)-clustering that simultaneously maximises α and minimizes ϵ.

3.2.1 Spectral Clustering

Not surpisingly, this optimization problem is NP-hard. So we need to use approximation algorithms or heuristics. Spectral partitioning or the use of the eigenvectors of a graph to induce partitions, is a commonly used heuristic for graph partitioning and related problems [11]. This method finds the Fiedler vector[1] [16] of the Laplacian matrix[2], **L** and partitions the vertices according to whether their value in this vector is less than or greater than a *splitting value* s. Popular choices for the splitting value s are: *bisection cut*, in which s is the median of $\{u_{21}, u_{22}, \dots, u_{2n}\}$; *ratio cut*, in which s is the value that gives the best conductance; *sign cut*, in which s is equal to 0; and *gap cut*, in which s is a value in the largest gap in the sorted list of Fiedler vector components [33].

[1]The eigenvector, $\vec{u_2} = (u_{21}, u_{22}, \dots, u_{2n})$ of the second smallest eigenvalue, λ_2

[2]The Laplacian matrix of a graph is derived from its weight matrix: $l_{ij} = -a_{ij}, l_{ii} = \sum_j a_{ij}$

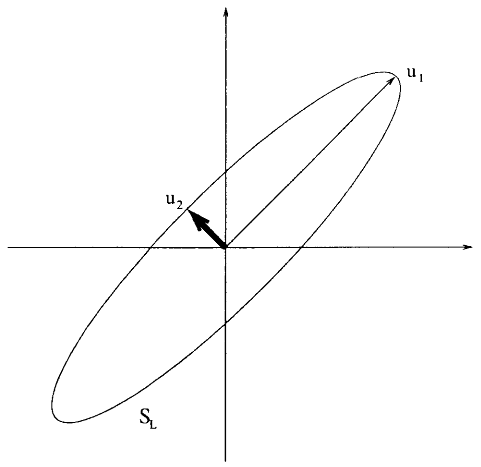

Figure 3-5: The vector $\vec{u_2}$ is the longest vector on $S_L \perp \vec{u_1}$ (This figure is not entirely accurate since $\lambda_1 = 0$ for a Laplacian matrix, hence the length of the major axis is actually ∞)

Figure 3-5 shows the geometric intution behind the use of the Fiedler vector. It turns out that the *relaxed* version of the minimum expansion cut problem is geometrically equivalent to finding the farthest point, $\vec{u}$, on the Laplacian ellipsoid[3], S_L, subject to the constraint that the components of $\vec{u}$ sum up to zero. The axes of such an ellipsoid are the eigenvectors of L with the lengths of axes inversely proportional to the eigenvalues. Also, the constraint on $\vec{u}$ is equivalent to stipulating that $\vec{u}$ be orthogonal to the vector of all ones ,ie, $u^T e = 0$ where $e = (1, 1, \ldots, 1)$. Using the fact that e is the eigenvector of the smallest eigenvalue ($\lambda_1 = 0$) for **L**, it is not hard to see that $\vec{u} = \vec{u_2}$ leads to the optimum value. The various splitting heuristics are nothing but rounding techniques to get an integral solution from the relaxed solution.

Kannan et al [25] incorporate this partitioning method into a "one-at-a-time" spectral algorithm for solving the clustering problem. The algorithm uses a parameter α^* which is the desired minimum conductance. The objective is to produce a clustering that minimizes ϵ. The algorithm is summarized on the next page.

Implementing the algorithm with $\alpha^* = \frac{\alpha}{6 log \frac{n}{\epsilon}}$ gives us the following worst case guarantee:

Theorem 3.2.1 (Kannan-Vempala-Vetta) *If A has a (α, ϵ)-clustering, then SPECTRAL CLUSTERING will find an $(\frac{\alpha^2}{36 log^2 \frac{n}{\epsilon}}, 10\sqrt{\epsilon} log \frac{n}{\epsilon})$-clustering.*

[3]The ellipsoid generated by the eqn $x^T L x = 1$

```
SPECTRAL CLUSTERING
{
  Input:  $\mathbf{L}$, the Laplacian similarity matrix for the dataset
  Output:  clusters of the dataset

  compute the Fiedler vector, $\vec{u_2}$ of $\mathbf{L}$.
  find the best ratio cut wrt $\vec{u_2}$.
  IF the conductance of the cut is $\geq \alpha^*$ then
    output the cluster.
  ELSE
    recurse on the induced pieces.
}
```

The parameter α controls the granularity of the generated clustering, a low value implies a coarse-grained clustering while a high value implies the opposite. The right value to choose depends on the particular problem domain and the kind of clustering desired. In our experiments, we have implemented this algorithm with $\alpha = 0.8$.

3.2.2 Evaluating the Quality of a Clustering

In order to evaluate the applicability of this algorithm for our text clustering problem, we need to come up with a measure that compares a given clustering with the "ground-truth" clustering. Intuitively, two clusterings are similar if each cluster in one is similar to some cluster in the other and vice-versa. Given two clusterings $C = C_1, C_2, \ldots, C_k$ and $C' = C'_1, C'_2, \ldots, C'_l$, we can come up with such a quantitative measure as follows. We define the similarity between two clusters C_i and C'_j by:

$$Sim(C_i, C'_j) = 2\frac{|C_i \cap C'_j|}{|C_i| + |C'_j|} \tag{3.5}$$

This quantity is 1 if and only if C_i and C'_j are identical, otherwise it is < 1. This measure allows us to find the closest match to a given cluster – it is simply the cluster which maximises this quantity. This leads us to the following similarity measure for clusterings C and C':

$$Sim(C,C') = \frac{1}{2}\left(\frac{1}{k}\sum_i \max_j Sim(C_i,C'_j) + \frac{1}{l}\sum_j \max_i Sim(C_i,C'_j)\right) \qquad (3.6)$$

A similar measure has been used in [18], although their measure is not symmetric with respect to the two clusterings.

3.3 Compression for Clustering

The key insight we had for solving the source based document clustering problem was to use compression for generating the similarity metric between documents. In particular, a lower cross-entropy between two documents suggests greater similarity between their sources and thus the cross-entropy numbers make for a meaningful similarity metric. Figure 3-6 shows a sample matrix of raw cross-entropies.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1	–	2.990	3.015	3.245	2.984	2.998	3.055	3.555	3.399	3.499	3.454	3.523	3.381
2	2.913	–	2.955	3.190	2.963	2.970	3.011	3.427	3.327	3.399	3.374	3.403	3.327
3	2.985	2.974	–	3.158	3.029	3.036	3.094	3.546	3.468	3.569	3.531	3.537	3.451
4	2.778	2.767	2.733	–	2.786	2.792	2.869	3.231	3.186	3.318	3.317	3.191	3.235
5	2.891	2.903	2.980	3.150	–	2.913	2.946	3.419	3.342	3.392	3.384	3.429	3.335
6	3.073	3.116	3.121	3.303	3.068	–	3.128	3.591	3.443	3.518	3.563	3.576	3.436
7	2.898	3.044	3.077	3.330	2.956	3.006	–	3.552	3.344	3.492	3.504	3.551	3.397
8	3.284	3.290	3.372	3.528	3.284	3.317	3.320	–	2.932	2.958	2.974	2.742	3.017
9	3.112	3.137	3.245	3.360	3.155	3.134	3.174	2.909	–	2.898	2.926	2.926	2.927
10	3.381	3.387	3.451	3.707	3.382	3.390	3.401	3.192	3.096	–	3.108	3.312	3.181
11	3.312	3.315	3.409	3.637	3.358	3.390	3.420	3.202	3.020	3.047	–	3.228	3.138
12	3.073	3.095	3.186	3.290	3.100	3.116	3.147	2.635	2.791	2.866	2.822	–	2.699
13	3.109	3.128	3.187	3.316	3.127	3.134	3.170	2.804	2.895	2.970	2.993	2.734	–

Figure 3-6: Sample matrix of raw cross-entopies. The ground truth clustering is indicated on the top of the table while the clustering produced by the spectral algorithm is shown on the left.

Since a lower cross-entropy corresponds to higher similarity, the actual similarity matrix **A** is generated by setting $a_{ij} = C - I_j(i)$, where C is a constant and $I_j(i)$ is the cross-entropy between documents i and j (Section 2.1). This is an *ad hoc* choice; one might very well argue for the usage of reciprocals or some other decreasing function of the cross-entropies. Also, the matrix is made symmetric by setting $a_{ij} = \frac{1}{2}(a_{ij} + a_{ji})$. The overall flow of the method is outlined on the next page.

```
DOCUMENT CLUSTERING
{
  Phase I
  FOR each document $D_i$ DO
    run compression *adaptively* on it to generate its model $M_i$
  FOR each document $D_i$ DO
    FOR each model $M_j$ DO
      run compression *statically* to compute the cross-entropy $I_j(i)$

  Phase II
  generate similarity matrix from cross-entropies
  run clustering on the similarity matrix
}
```

3.4 Experimental Results

3.4.1 Clustering based on Authorship

We performed clustering experiments on randomly generated groupings of documents in our corpus of classical texts. The clusterings generated by the spectral clustering algorithm were compared with the ground truth using the criterion mentioned in Subsection 3.2.2.

Table 3.1 tabulates the results of this experiment.

Table 3.1: Clustering quality for different compression methods

Method A	Method B	Method C
0.69	**0.74**	0.59

Method B again outperforms the other methods although in this case, one cannot draw too general a conclusion from the quality numbers since they do not have as clean an interpretation as the accuracy percentages in the previous chapter.

Figure 3-7 shows the variation of clustering quality with document size. Interestingly, the curve stays relatively flat even upto document sizes of 500 bytes.

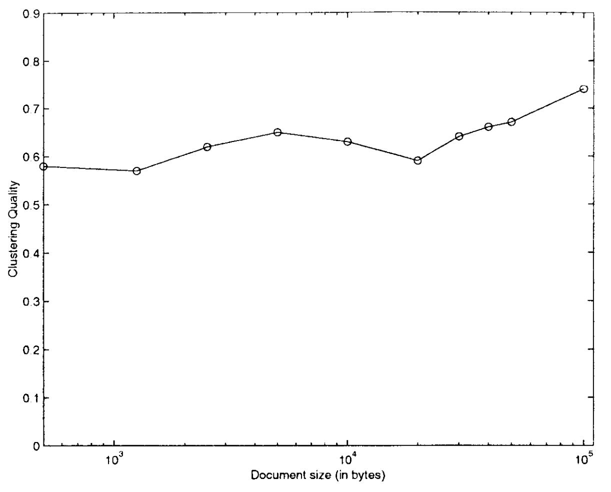

Figure 3-7: Clustering quality versus document size

3.4.2 Clustering based on Language

We performed clustering on the same set of languages used for the supervised language classification experiment. Clustering on this dataset yielded a quality measure of 0.86 for method A and 0.92 for method B.

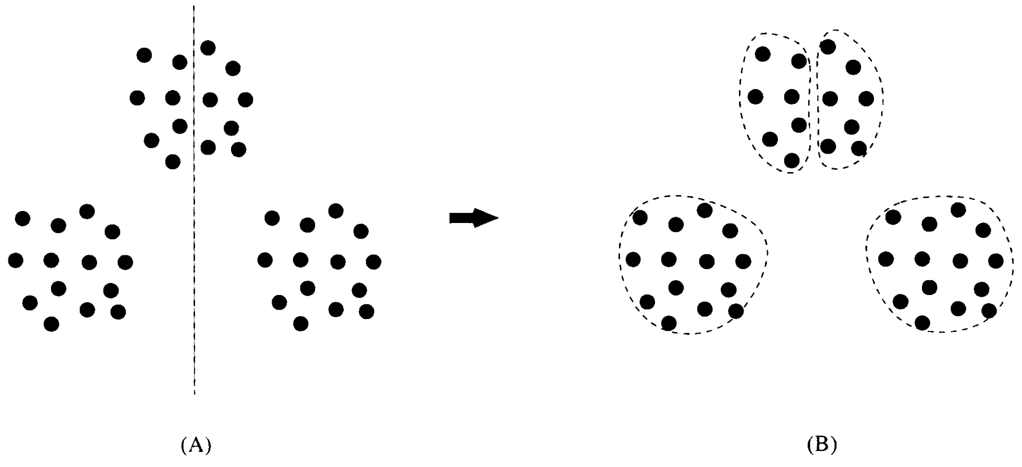

Figure 3-8: An example of a bad clustering produced by the spectral algorithm. A shows the initial bisection cut and B shows the final clustering

This experiment revealed some drawbacks of the spectral clustering technique. Since the algorithm chooses the cut that minimizes the conductance, it tends to favor cuts that divide the vertex set into roughly equal portions. This might be undesirable in a situation such as the one shown in Figure 3-8. In this example, there are 3 well-defined clusters but the first cut ends up slicing one of the clusters. The algorithm ends up generating the clustering shown on the right, which is clearly not the desired one. One could conceivably solve this problem by adding a post-processing phase to the spectral algorithm that merges clusters as long as the merged cluster has the minimum desired conductance. It might be worthwhile to investigate whether this extra processing actually improves the theoretical performance guarantee of the spectral algorithm. In our implementation, we do not perform this extra computation; instead we use an *ad hoc* heuristic that chooses one of the ratio cut or sign cut methods (Section 3.2.1) to find the bisecting cut, based on the spread of values in the Fiedler eigenvector.

3.4.3 Hierarchial Classification

Supervised classification of a given document takes time that is linear in the number of categories. This can be a bottleneck if the number of categories is large. A hierarchial approach can be helpful in such a case; it can potentially reduce the number of compression runs to a factor logarithmic in the number of categories (although with an accompanying space overhead). This can be done by clustering the categories into a hierarchial tree and performing classification by following a branch of the tree to its leaves. The training set for each node of the tree is obtained by combining the training sets of all the nodes below it. Figure 3-9 shows the classification tree for the language identification problem described in the previous section.

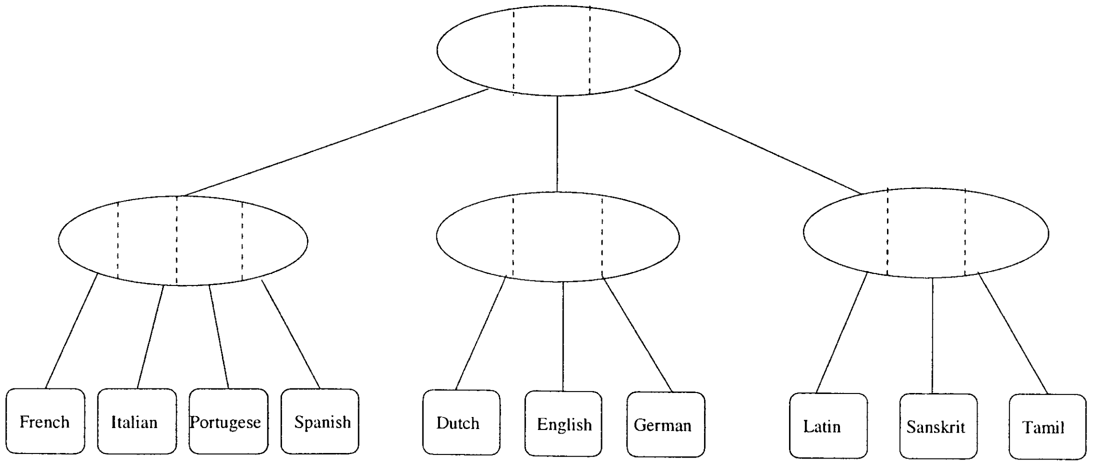

Figure 3-9: Classification tree for languages

The language groupings generated by the clustering algorithm, are interesting in their own right. Although the first two groups correspond to the well-studied *Italic* and *Germanic* subfamilies in the Indo- European family of languages [3], the grouping together of Sanskrit and Latin (both Indo-European languages) with Tamil (a Dravidian language) is somewhat surprising. Such a compression based technique could actually be used as an aid by linguists trying to discover similarities and dissimilarities between classes of languages.

Chapter 4

Conclusions

We have experimented with the idea of using compression as a means to achieve source based text classification. Among the different compression techniques we tried, we found character based PPM to be the most effective. The results for the language identification problem indicate that the method is highly reliable even for small document sizes. The corresponding results for the author identification problem show that the method has a high rate of identification accuracy for large document sizes but it degrades significantly for document sizes below a certain threshold. Thus, this technique can only be used in application domains with large enough text corpora. In particular, it cannot be used as is, for our original motivating problem – verifying users by the testimonials they write (although, it can definitely be used as an adjunct with other means).

In our opinion, the applicability of this technique can be extended to any domain where the language of discourse can be modelled by a probabilisitic generation of symbols from a finite alphabet. An interesting possibility relates to the domain of musical compositions, which like texts, obey structural constraints that make probabilistic modelling feasible.

Compression also seems promising for the copy detection problem although more work needs to be done to evaluate the pros and cons of the technique. In particular, one needs to evaluate the robustness of this technique in the face of deliberate changes made to the copied text by a malicious adversary, in order to fool the compression technique.

In this work, we have mainly focussed on demonstrating a "proof-of-concept". We have mostly ignored the issues of space and time efficiency. For the clustering problem, the primary bottleneck is the construction of the similarity matrix, requiring $\Theta(n^2)$ compression runs. It would be desirable to come up with an algorithm that achieves good clustering without looking at all the entries in the similarity matrix. Drineas et al [12] have proposed a sampling based approach that appears promising for large clustering problems.

It would be interesting to see whether the entropy numbers themselves can be estimated by a sampling approach. Alon et al [4] describe a *property testing* technique for the membership problem for regular languages, which tests whether a string is accepted by a finite state automaton or is "far" from any such string, by looking at only a constant number of places in the string. Extending this idea to Markov models would be highly desirable as it would enable the estimation of entropy without examining the entire document.

Appendix A

Relevant Code and Data

The code and data used for the experiments described in this thesis can be accessed from `http://supertech.lcs.mit.edu/~nitin/classify`. The interested reader should be able to adapt the code to her needs from the accompanying documentation.

The compression algorithms described in this thesis are by no means exhaustive. Notable absentees are the Burrows-Wheeler transform and grammar based compression techniques. The reader interested in trying out other methods for this problem, can access source code for a variety of compression techniques from `http://www.dogma.net/DataCompression/SourceCode.shtml`.

Gaining access to digital texts can be problematic owing to copyright issues. A useful starting point for accessing texts in the public domain is the Gutenberg online library project (`http://www.gutenberg.org`).

Appendix B

List of works

Table B.1: List of authors and their works

Author	Works
Jane Austen	Pride and Prejudice, Emma Mansfield Park, Sense and Sensibility
Daniel Defoe	Robinson Crusoe, From London to Land's End Moll Flanders, Tour through the Eastern Counties of England
Charles Dickens	A Tale of Two Cities, A Christmas Carol Oliver Twist, Great Expectations
R. W. Emerson	Essays I, Essays II, English Traits, The Conduct of Life
Rudyard Kipling	Kim, Plain Tales from the Hills Captains Courageousi, American Notes
W. Shakespeare	Hamlet, The Merchant of Venice Romeo and Juliet, Sonnets
G. B. Shaw	Misalliance, You Never Can Tell An Unsocial Socialist, Man and Superman
Mark Twain	Huckleberry Finn, A Connecticut Yankee in King Arthur's Court Life on the Mississipi, Following the Equator
H. G. Wells	The First Men in the Moon, The War in the Air The New Machiavelli, Dr. Moreau
Oscar Wilde	Lord Arthur Savile's Crime and Other Stories, Essays and Lectures The Picture of Dorian Gray, An Ideal Husband

Bibliography

[1] `http://cluster.cs.yale.edu`.

[2] `http://www.speech.cs.cmu.edu/speech/SLM_info.html`.

[3] *Columbia Encyclopaedia, 6th ed.* Columbia University Press, 2000.

[4] N. Alon, M. Krivelevich, I. Newman, and M. Szegedy. Regular languages are testablewith a constant number of queries. In *Proc. 40th IEEE Symp. on Foundations of Comp. Science*, 1999.

[5] Guy E. Blelloch. Introduction to data compression. Course notes for: Algorithms for the real world, 2000.

[6] J. Brassil, S. Low, N. Maxemchuk, and L. O'Gorman. Electronic marking and identification techniques to discourage document copying, 1994. Technical Report AT&T Bell Laboratories.

[7] M. Charikar, S. Guha, D. Shmoys, and E. Tardos. A constant-factor approximation for the k-median problem. In *Proc. 31th ACM Symp. on Theory of Computing*, pages 1–10, 1999.

[8] Stanley F. Chen. Building probabilisitic models for natural language. Ph.D. Thesis, Harvard University, 1996.

[9] A. Choudhury, N. Maxemchuk, S. Paul, and H. Schulzrinne. Copyright protection for electronic publishing over computer networks, 1994. Technical Report AT&T Bell Laboratories.

[10] J. G. Cleary and I. H. Witten. Data compression using adaptive coding and partial string matching. *IEEE Transactions on Communications*, 32(4):396–402.

[11] W. E. Donath and A.J. Hoffman. Algorithms for partitioning of graphs and computer logic based on eigenvectors of connection matrices. *IBM Technical Disclosure Bulletin*, 15:938–944, 1972.

[12] P. Drineas, Alan Frieze, Ravi Kannan, Santosh Vempala, and V. Vinay. Clustering in large graphs and matrices. In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, 1999.

[13] S. Dumais, J. Platt, D. Heckermann, and M. Sahami. Inductive learning algorithms and representations for text categorization. In *Proc. Intl. Conf. on Info. and Knowledge Management*, pages 148–155, 1998.

[14] M. E. Dyer and A. Frieze. A simple heuristic for the p-center problem. *Oper. Res. Let.*, 3:285–288, 1985.

[15] J.M. Farringdon. *Analysing for Authorship: A Guide to the Cusum Technique*. University of Wales Press, 1996.

[16] M. Fiedler. A property of eigenvectors of nonnegative symmetric matrices and its applications to graph theory. *Czechoslovak Mathematical Journal*, 25(100):619–633, 1975.

[17] E. Frank, C. Chui, and I.H. Witten. Text categorization using compression models. In *Proc. of the Data Compression Conference*, 2000.

[18] M. Gavrilov, D. Anguelov, P. Indyk, and R. Motwani. Mining the stock market: which measure is best ? *KDD*, 2000.

[19] D.I. Holmes. The evolution of stylometry in humanities scholarship. *Literary and linguistic computing*, 13:111–117, 1998.

[20] David Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the IRE*, volume 40, pages 1098–1101, 1952.

[21] Piotr Indyk. A sublinear time approximation scheme for clustering in metric spaces. In *Proc. 40th IEEE Symp. on Foundations of Comp. Science*, pages 154–159, 1999.

[22] A. K. Jain and R.C. Dubes. *Algorithms for clustering*. Prentice-Hall, 1988.

[23] K. Jain and V. Vazirani. Primal-dual approximation algorithms for metric facility location and k-median problems. In *Proc. 40th IEEE Symp. on Foundations of Comp. Science*, pages 2–13, 1999.

[24] T. Joachims. Text categorization with support vector machines: learning with many relevant features. LS-8 Report 23, Computer Science Department, University of Dortmund.

[25] Ravi Kannan, Santosh Vempala, and Adrian Vetta. On clusterings: good, bad and spectral. In *Proc. 41th IEEE Symp. on Foundations of Comp. Science*, 2000.

[26] K. Lari and S.J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[27] D. D. Lewis and M. Ringuette. A comparison of two learning algorithms for text categorization. In *Proc. Annual Symposium on Document Analysis and Information Retrieval*, pages 37–50, 1994.

[28] A. Moffat. Implementing the PPM data compression scheme. *IEEE Transactions on Communications*, 38(11):1917–1921, 1990.

[29] Mark Nelson. LZW data compression, 1989. Dr Dobb's Journal `http://dogma.net/markn/articles/lzw/lzw.htm`.

[30] H. T. Ng, W.B. Goh, and K.L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. In *Proc. ACM SIGIR*, pages 67–73, 1997.

[31] C.E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:623–656, 1948.

[32] N. Shivakumar and H. Garcia-Molina. A copy detection mechanism for digital documents. In *Proc. of 2nd International Conference in Theory and Practice of Digital Libraries, Austin, Texas*, 1995.

[33] D. Spielman and S. Teng. Spectral partitioning works: planar graphs and finite element meshes. In *Proc. 37nd IEEE Symp. on Foundations of Comp. Science*, 1996.

[34] W. J. Teahan. Modelling english text. DPhil thesis, Univ. of Waikato, N.Z., 1997.

[35] W. J. Teahan. An improved interface for probabilistic models of text, 2000.

[36] W. J. Teahan. Text classification and segmentation using minimum cross-entropy. In *International Conference on Content-Based Multimedia Information Access (RIAO)*, 2000.

[37] F.J. Tweedie, S. Singh, and D.I. Holmes. Neural network applications in stylometry: The federalist papers. *Computers and the humanities*, 30:1–10, 1996.

[38] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential date-compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.

[39] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.