INFOSIFT: ADAPTING GRAPH MINING TECHNIQUES FOR DOCUMENT CLASSIFICATION

The members of the Committee approve the master's thesis of Manu Aery

Sharma Chakravarthy
Supervising Professor _____

Diane Cook _____

Alp Aslandogan _____

To my Family

INFOSIFT: ADAPTING GRAPH MINING TECHNIQUES FOR DOCUMENT CLASSIFICATION

by

Manu Aery

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE AND ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2004

ACKNOWLEDGEMENTS

I express my sincere gratitude to my supervising professor, Dr. Sharma Chakravarthy who has been a great motivating factor and a constant source of encouragement throughout my masters' research. Without his guidance and excellent foresight this thesis would have only remained a great idea. I thank Dr. Diane J. Cook for patiently replying to my numerous queries and Dr. Alp Aslandogan for introducing me to the exciting field of Text Mining. I am sincerely thankful to them for serving on my committee.

I thank the Almighty for the infinite grace and benevolence. My family has been most supportive, but for their love and encouragement the demands of research would have been hard to meet. In my interactions with Dr. Lynn Peterson through the course of my graduate career, I sincerely thank her for her patience and understanding.

I would like to express my appreciation to Raman Adaikkalavan, Laali Elkhalifa, Sridhar Reddy Varakala, Ramanathan Balachandra, Srihari Padmanabhan and other friends in ITLAB. I am grateful to Shravan Chamakura for a maintaining a well-administered research environment. Sincere thanks are due to Mr. Patrick McGuigan for the processing facilities provided in the high performance cluster. I also greatly appreciate my numerous other friends for their love and support.

This work was supported, in part by NSF (grants IIS-0097517 and IIS-0326505).

18 November 2004

ABSTRACT

INFOSIFT: ADAPTING GRAPH MINING TECHNIQUES FOR DOCUMENT CLASSIFICATION

Publication No. _____

Manu Aery, M.S.

The University of Texas at Arlington, 2004

Supervising Professor: Sharma Chakravarthy

Text classification is the problem of assigning pre-defined class labels to incoming, unclassified documents. The class labels are defined based on a sample of pre-classified documents, which are used as a training corpus. A number of machine learning, probabilistic, and information retrieval based approaches have been proposed for text classification. The same have also been applied to solve the problem of email and web page classification. While most of these techniques rely on extracting keywords or highly frequent words for classification, they ignore the importance of extracting a group of related terms that co-occur and are unable to capture relationships between words. These patterns of term association are important, as there is reason to believe that documents within a class adhere to a set of patterns, and that these patterns closely correspond to, and are derived from the documents of the particular class.

A classification system that determines the patterns of various term associations that emerge from documents of a class, and uses these patterns for classifying similar documents is needed. This thesis proposes a novel graph-based mining approach for document classification. Our approach is based on the premise that representative – common and recurring – structures or patterns can be extracted from a pre-classified document class and the same can be used effectively for classifying incoming documents. To the best of our knowledge, there is no existing work in the area of text, email or web page classification based on pattern inference and the utilization of the learned patterns for classification. A number of factors that influence representative structure extraction and classification are analyzed conceptually and validated experimentally. In our approach, the notion of inexact graph match is leveraged for deriving structures that provide coverage for characterizing the contents of a document class. The ability to classify based on similar and not exact occurrences is singularly important in most classification tasks, as no two samples are exactly the same. Extensive experimentation validates the selection of parameters and the effectiveness of our approach for text, email and web page classification.

The novel idea proposed in the thesis aims at establishing the ground work for adapting graph mining techniques for various classification problems, not necessarily limited to text.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Management of data and information has always been concerned with the need for extracting important and relevant aspects of a document and storing it for efficient retrieval later. This need is highlighted in various ways, cataloging of books in a library to enable quick lookup for a book of interest, the index of term usages in a book to easily locate relevant information and so on. While these deal with management of information on a small scale, with the advent of the World Wide Web it has become important to look beyond the notion of 'indices' to retrieve relevant information. As the Web can be viewed as a huge information repository, determining what is relevant in such a vast storehouse is exceedingly important if we are to effectively manage the information that is available and store it in a manner that expedites later retrievals.

Information management is of great significance today as information technology has revolutionized the way data and knowledge is shared among users spread over different parts of the globe. Instant access to large amounts of information available through the Internet entails a need for mechanisms that determine the relevance of information being accessed. The task can be to simply filter information based on the presence or absence of certain keywords. In other cases, it may be required to take into account the context of term appearances to ensure extraction of information that is relevant to the particular desired sense of the term. For example, if the intent is to extract all information regarding the programming language Java, providing only the keyword 'java' may result in a lot of irrelevant information. The problem can be circumvented by providing additional information for querying purposes or processing the extracted

information to determine what is relevant. In other cases, the management of information maybe as complicated as generating a summary of the extracted information. Another mechanism for information management is classification, which allows one to separate information belonging to different categories of interest.

1.1 Classification: Overview

The problem of classification involves the process of learning relevant features or attributes of a class and using the same to determine if a new sample belongs to that class. Pre-classified examples in classes are used as a training set to build a descriptor for each class. To determine the destination class for an unknown sample, it is compared with the descriptors of all classes and categorized where the similarity is maximal. A practical scenario would be a consumer company seeking to maximize sales of a new product. User behavior corresponding to a class of customers, who in the past have availed such offers can be learnt to derive relevant attributes. Subsequently, spending patterns of new customers can be compared with what has been learnt to determine if they are potential customers for target marketing. Of course, the scope of classification extends to a host of other applications. For the management of information, which is the task at hand, classification allows us to separate information belonging to different classes. This arrangement, wherein all relevant information that corresponds closely is categorized together presents many benefits. It allows us to retrieve similar objects easily, as they are grouped together and also it does enable us to search effectively for a particular sample.

This thesis aims at applying a novel approach based on graph mining to solve the problem of classification. As the particular domain of application, we have chosen to work with text. Text in our case, encompasses general text, emails and web pages. We have studied the feasibility of our approach and established our claims for adapting this

novel approach. In the sections that follow, we will discuss in detail the various aspects of text, email and web page classification, the issues present and the inherent challenges of the domain.

1.2 Text Classification

Text classification is the problem of assigning pre-defined class labels to incoming, unclassified documents/text[1]. The class labels are defined based on a sample of pre-classified documents used as a training corpus. Text classification has traditionally been applied to documents in the context of information retrieval and a large body of work on classification is available. In this thesis, we address the problem of classifying general text, emails and web pages. While all of them deal with classifying unknown documents into known classes, the domains of electronic mail (or email) and web pages provide certain domain knowledge that can be incorporated into the classification task. We have considered the discussion of each as a separate topic.

Text classification, especially for categorizing news stories has prompted a range of solutions that draw upon techniques from various fields. These include machine learning techniques such as Support Vector Machines (SVM) [1], decision tree classifiers [2, 3, 1], k-Nearest-Neighbor algorithms (kNN) [4, 5, 6], and neural networks [7, 8]. Statistical techniques such as Linear Least Square Fit [9], the probabilistic Bayesian classification [10, 11, 12, 3, 1, 13] and rule induction [14, 15, 16, 17] are few of the other approaches used for classifying text. Also, the class of Term Frequency Inverse Document Frequency (TF-IDF) classifiers [18] from information retrieval have been applied to the problem of text classification.

All text classification approaches extract relevant features from the documents that form the training set for the class. These features are used to either build a vector that

[1]The two terms have been used interchangeably throughout

consists of the extracted terms qualified by their occurrence frequencies or probability scores for classification (in case of the TF-IDF, Naive Bayes, k-NN and others) or are used to train the classifier to learn those features corresponding to the class to enable classification (as in decision trees, neural networks, SVMs and so on). The various text classification techniques will be considered in detail when the related work is discussed in the next chapter. We will for the time being delimit ourselves to the problem of email classification, as the challenges of this domain are not often found in the context of general text classification and the methods applied to solve the same.

1.3 Email Classification

In the Internet age, rapid dissemination of data and a quick method to apprise others of the information available is possible by means of electronic mail. Electronic mail can be viewed as a special type of document as it is primarily text along with some identifying information unique to it (e.g., from, to, subject, cc, attachments etc.) Email has evolved as a convenient means of communication between individuals as well as groups. It is a fast, efficient and an inexpensive method of reaching out to a large number of people at the same time. This very reason is also the bane of electronic mail communication. Most users are often overwhelmed by the sheer volume of email sent and received. Users often find themselves expending large amounts of time and effort sifting through the mass of email messages and classifying them to their corresponding folders [19]. A tool (hopefully automated) is needed for the management of email due to its sheer volume and the need to retrieve relevant emails as needed later. Mis-classification[2]

[2]Classification of emails can be compared to the cataloguing of books in a library. A book is assigned a call number (context of the email in our case) and assigned to the appropriate shelf (folder in our case). If the book (email) is misplaced in a wrong shelf (folder), its retrieval will be extremely difficult; no wonder ordinary library users are not asked to re-shelve a book.

and putting it in the wrong place is as good as losing the email as it will be extremely difficult to locate later. One aspect of email management is to classify email messages into appropriate folders automatically with acceptable accuracy. An automated technique for classifying email seems even more important considering the amount of time spent in processing emails by individuals. This time can be greatly reduced if either traditional classification techniques can be adapted or new techniques developed to address the problem of email classification, thereby automating the process with a good amount of accuracy and efficiency. Certainly, the presence of an automated system can reduce the time to classify mails, search for a particular mail and retrieve relevant mails. In general, any email management system would require a classification component for the effective management of emails. Once this is done, indexing (analogous to the library catalog) can also be accomplished using the information gleaned while performing classification. A word or phrase-based index can be created to locate folders and emails containing the words. Hence, an email classifier is one of the critical tools needed for the effective management of information exchanged in the Internet age.

1.3.1 Challenges

Although email classification can be viewed as a special case of text classification, the characteristics of documents and emails differ significantly and as a consequence email classification poses additional challenges, not often encountered in text or document classification [20]. Email classification is trickier than text classification as it is based on personal preferences (e.g., different mail filing habits ranging from those who rarely classify emails to those who maintain a strict hierarchy), varying criteria for filing emails into folders and so on. Consequently, email classification uses disparate criteria which are difficult to quantify. Also, documents are richer in content as compared to emails whose content can vary dramatically from folder to folder, hence conventional approaches may

not be well-suited. In addition, as opposed to a static set of corpus typically used for training in text classification, the email environment is constantly changing, with a need for adaptive and incremental re-training. Some of the differences and the concomitant challenges are outlined below:

1. Manual classification of emails is based on personal preferences and hence the criteria used may not be as simple as those used for text classification. For example, different users may classify the same message into vastly different folders based on their preference. This varies significantly from document classification where the class label associated with a document is independent of the user. This distinction needs to be taken into account by any technique proposed/used for email classification.
2. Each users' mailbox is different and constantly evolving. Folder contents vary from time to time as new messages are added and old messages are deleted. A classification scheme that can adapt to varying folder characteristics is important.
3. The information content of emails vary significantly, and other factors, such as the sender, group the email is addressed to, play an important role in classification. This is in contrast to documents which are richer in content resulting in easier identification of topics or context. In case of emails, the above factors assume importance as the body of the message may not yield enough information. Also, most emails do not follow a fixed or standard vocabulary rendering the use of a lexical thesauri difficult.
4. The characteristics of folders may vary from dense (more number of emails) to relatively sparse. A classification system needs to perform reasonably well even in the absence of a large training set. A graceful degradation in accuracy may be acceptable with decreasing training data.

5. Emails within a folder may not be cohesive i.e., the contents may be disparate and not have many common words or a theme. We characterize these folders on a spectrum of homogeneous to heterogeneous. A folder may lose its homogeneity as it becomes dense making it difficult to associate appropriate central theme/structures with the folder.
6. Emails are typically classified into subfolders within a folder. The differences in the emails classified to subfolders may be purely semantic (e.g., individual course offerings within the courses folder, travel within the projects folder etc.,) or theme oriented. The ability to classify emails to appropriate subfolders will require a clear separation of representative folder characteristics or traits. Email folders may also be split when the number of emails in the folder becomes unmanageable or contents of many folders may be merged at times. Any approach used for email classification should be able to deal with these nuances which are typically absent in text classification.

Text classification techniques can be applied to solve the problem of email classification, but they have to be adapted to take into account the differences listed above. Additionally, they can draw information available in the email domain for classification. A number of text classification techniques have been applied to the problem of email classification. In *Re:Agent* [21] Boone uses a machine learning based technique, the k-Nearest Neighbor algorithm. Rule based classification [22], Naive Bayesian probability based classification [23] and the class of term-frequency, inverse frequency (TF-IF) classifiers widely used in information retrieval have been used in email classification systems [21, 24]. We will further elaborate on these in the related work section.

1.4 Problem Statement

Most of the techniques mentioned above rely on extracting keywords or highly frequent words for classification. They ignore the importance of extracting a group of related terms that co-occur. There is no reason to believe that documents/emails within a class/folder adhere to a set of patterns and that these patterns closely correspond to, and are derived from the documents or emails of the particular class or folder. A classification system that determines the patterns of various term associations that emerge from documents/emails of a class/folder and uses these patterns for classifying similar unknown samples is needed. The ability to classify based on similar and not exact occurrences is singularly important in most classification tasks, as no two samples are exactly alike. To the best of our knowledge, there exists no work in the area of text, email, or web page classification that infers patterns from text/emails and relies on these learnt patterns for classification.

The belief that the process of classification or supervised learning, which entails grouping related or similar entities, can benefit from the application of data mining techniques has prompted this research direction. Data Mining is the process of discovering interesting, non-trivial, implicit, previously unknown and potentially important information and patterns from data [25]. There are many data mining techniques such as association rule mining [26], which discovers associations between items in a set of transactions, sequence pattern mining, which is the mining of frequent patterns related to time or other sequences, graph mining [27], which is the discovery of interesting patterns/subgraphs in an input graph, frequent subgraphs (or FSG) [28] in which a frequent subgraph is identified in a large database of graphs and so on.

The interesting patterns or knowledge uncovered by the discovery process can be used effectively in a decision making process. As data mining is the process of pattern discovery and we believe that the documents/emails in a given class/folder exhibit similar

patterns, the process of discovering these patterns can be automated by the use of data mining techniques, and the discovered patterns can later be used for the process of classification. The main motivation behind this thesis is to investigate the applicability of data mining techniques and adapt them, where appropriate, for pattern discovery in text/emails and to subsequently use these patterns for classification. We believe that text and web documents have a structure in the form of the title, keywords, section headings, the HTML tag elements in case of web pages and the document body. Also, emails can be represented using structural relationships as an email message has a structure in the form of the information contained in the headers, the subject and the body. This is radically different from the existing corpora of work where emails are considered as general text having no particular structure. By attaching structural information to text and emails they can be made amenable to graph mining.

1.5 Our Approach

In this thesis we propose a novel approach that adapts graph mining techniques for document, email and web page classification [29]. The approach is based on the premise that representative – common and recurring – structures/patterns can be extracted from pre-classified documents in a class or emails in a folder and the same can be used effectively for classifying unknown document samples or incoming email messages. Supervised learning along with domain characteristics are exploited to identify the composition of previously labelled documents or emails and these are used for the classification of unknown text samples or incoming emails.

Documents in a given class correspond to one another and the similarity between them provides the discriminating capability required to distinguish one class from another. Also, users organize email folders based on their content, certain patterns that occur in the email messages, and personal preferences (for creating folders and sub-folders).

Our approach is based on the premise that a class or an email folder consists of representative documents or emails and the structure and content of these representative documents and emails can be extracted by adapting graph-based mining techniques to work with domain knowledge. We also hypothesize that the notion of *inexact graph matching* (or isomorphic graph comparison) is critical for this to work, as it helps in grouping *similar* structures within documents and emails instead of looking for exact/identical matches that may be difficult to find in the textual domain.

Briefly, in our approach, a document class or email folder is mined to obtain frequent and representative patterns using inexact graph matching guided by a threshold value. When an unknown sample is to be classified or a new email arrives, it is matched with the set of patterns for each class/folder. In case of a match, a classification rank is associated with the unknown sample or incoming email with respect to the class or the email folder. The actual classification is based on the rank (i.e., the document or email is classified into the category or folder for which it exhibits the highest rank).

Significant work carried out in the area of graph based data mining includes the frequent subgraph discovery algorithm (FSG) [28], the Subdue substructure discovery system [27] and the frequent graph miner:*gSpan* [30] among others. Our work requires a means of substructure discovery directed by specific constraints (explained later). The notion of matching inexactly within bounds dictated by various domain characteristics is necessary. Any of the existing techniques for substructure discovery can be employed in our work. FSG and gSpan do not have this notion of matching inexactly within a threshold value as they use canonical labelling. We have chosen to use the Subdue substructure discovery system as it supports many of the features that we consider are important. For example, inexact match is provided with a threshold value that can be varied to detect similar subgraphs which differ in edge and vertex labels by a specified maximum value. The Subdue [27] substructure discovery algorithm is used to discover frequent and rep-

resentative patterns for a given folder. Subdue discovers frequently occurring subgraphs using the minimum description principle [31]. Isomorphism or inexact graph match is used to make sure similar (and not merely exact) substructures are identified. In our approach, the notion of inexact graph match is leveraged for deriving structures that provide coverage for characterizing the contents of a class or email folder. Since Subdue identifies a large number of such patterns, they are sifted into a manageable number of patterns using several criteria such as, the frequency of occurrence, size of the pattern, average size of documents or emails in the class or email folder, the size of the document class or email folder and so on. The substructures are ranked and the set of representative substructures that best characterize the class or folder are chosen. Subsequently, an unknown document or incoming email message is compared with the representative set of substructures of each class/folder. Again, inexact graph match techniques are used to determine the destination class/folder for the unknown sample or email message. It is categorized to the class or email folder where the match is maximal. In case of email classification, if no match occurs, the email is classified to a default folder and user-intervention may be needed to classify it.

1.6 Contributions

The main contribution of this thesis is in the novelty of the approach in considering mining techniques for classification. Although mining techniques have been proposed earlier, to the best of our knowledge, this is the first attempt to assess the applicability of graph mining for classification. Another contribution is the evaluation of current mining approaches, their applicability to the problem at hand, and the adaptation of those approaches for text and email classification. Graph mining (or any other mining technique) does not perform classification, hence the problem is not straightforward. One of the primary contributions of this thesis is the formulation and use of graph mining

techniques to solve the problem of document classification. Finally, the last but not the least contribution of this thesis is an extensive evaluation of graph mining techniques and the effect of various parameters on classification. We analyze these parameters to infer their effects and provide experimental results to support the analysis. A number of factors that influence representative structure extraction and classification are analyzed conceptually and validated experimentally. We have adapted our approach to both general text, email and web page classification (using different graph representations as will be detailed later) to provide a well-rounded formulation of graph based data mining for all kinds of text. The approach presented in this thesis provides a general framework for using graph mining techniques and an approach to leveraging domain characteristics for classification.

Since we are dealing with document, email and web page classification into the associated document classes, email folders and web page categories, the terminology used throughout is 'text classification into classes' unless specified otherwise.

The remainder of the thesis is organized as follows. *Chapter 2* presents the related work in the areas of text, email and web page classification. *Chapter 3* gives an overview of graph based mining and a brief description of the Subdue substructure discovery system. *Chapter 4* presents an overview of the $InfoSift$ classification system. *Chapter 5* explains the system design along with the detailed discussion of the parameters involved. The implementation details, experimental results and comparisons along with the naive Bayesian technique are presented in *Chapter 6*. Conclusions and future work are outlined in *Chapter 7*.

CHAPTER 2

RELATED WORK

Text classification is the problem of assigning pre-defined class labels to incoming, unclassified documents. The class labels are defined based on a sample of pre-classified documents used as a training corpus. The problem of text classification is well researched and various methods proposed to solve the problem include machine learning techniques, probability based theorems and information retrieval approaches among others.

This chapter presents a brief overview of some of the conventional approaches applied to solve the problem of text classification. Some of these techniques have been used to address the problem of email and web page classification as well. The various techniques proposed for classification include Support Vector Machines (SVM) [1], decision trees [2, 3, 1], k-Nearest-Neighbor (k-NN) classifiers [4, 5, 6], Linear Least Square fit technique [9], rule induction [14, 15, 16, 17], neural networks [7, 8] and Bayesian probabilistic classification [10, 11, 12, 3, 1, 13]. Also, the class of Term Frequency – Inverse Document Frequency (TF-IDF) classifiers [18] from information retrieval have been applied to the problem of text classification. As is evident, the different techniques that address the problem have been drawn from very diverse fields.

In the sections that follow, a brief description of some of these text classification techniques is provided. Many email classification systems that apply these techniques to automate the task of filing emails have been developed. A discussion of some of these systems is presented when we consider the related work in the area of email classification. For classifying web pages, techniques that combine conventional text classification approaches and incorporate the domain knowledge have been proposed. Some of these

techniques are considered in detail when we present the relevant work in the area of web page classification. We will now discuss some of the text classification approaches that have been outlined earlier.

2.1 Text Classification Techniques

This section provides a brief overview of some of the popularly used text categorization techniques. The various text classification methods have been drawn from fields as diverse as machine learning, probability theory and statistical learning theory amongst others. To provide an idea of the diversity by way of which each of these methods solve a text classification problem at hand, we will now discuss some of the aforementioned techniques.

2.1.1 Support Vector Machines

Support Vector Machines were introduced by Vapnik [32] in 1979 [33], but have become popular in the last decade or so. Support Vector Machines belong to the set of discriminant classifiers (which include neural networks and decision trees among others). They are based on the Structural Risk Minimization principle [33] and aim at minimizing structural risk instead of empirical risk. Let us consider the simplest case that corresponds to a linearly separable vector space. The problem here is to find a decision surface that best separates the positive and negative examples of a class. A decision surface that does so is called a 'hyperplane' and includes the notion of a margin, which corresponds to how much the decision surface can be moved without affecting classification. A linear SVM can therefore be stated as a hyperplane that maximizes this margin between a set of positive and negative examples of a class. The margin is the distance from the hyperplane to the nearest of the negative and positive samples. The solid line in figure 2.1 shows the hyperplane that separates the positive and negative training samples of a class and

the thin lines on either side define the margin by which the hyperplane can be moved without causing misclassification. The hyperplane in the figure has maximal margin, any other decision surface will have a smaller margin than the one shown.

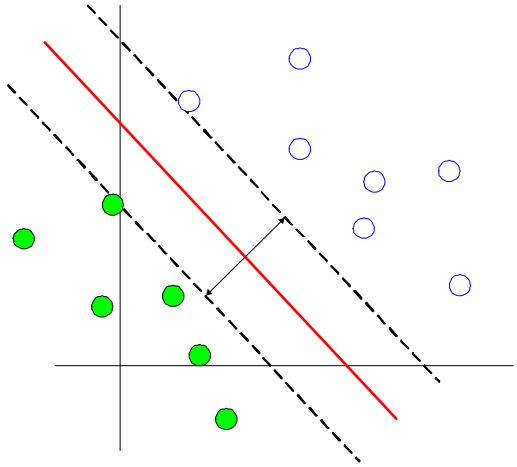

Figure 2.1 The decision line(solid) with maximal margin
The Support Vectors are points on the dashed lines

The optimal separating hyperplane is given by the equation

$$w * x - b = 0 \tag{2.1}$$

The linearly separable cases can be generalized to linearly non-separable cases. The performance of Support Vector Machines for text classification tasks has been studied in detail and they exhibit a remarkably better performance than most other classification techniques [34]. SVM classifiers perform well even in the presence of sparse data, as in effect the classification mainly depends upon the support vectors of a class. They are capable of handling large data sizes and the classifier performance is consistently good.

2.1.2 k-Nearest Neighbor Classifiers

In k-NN classifiers, as the name suggests, the classification of an unknown test sample is based on its 'k' nearest neighbors. The assumption is that the classification of an instance is based on others that are similar to it. Each document in the training set is represented by a feature vector, which is the set of relevant attributes of the sample. The technique for feature extraction can be as simple as the occurrence frequency of the term in the document. To classify an unknown sample, its corresponding feature vector is constructed and compared with the feature vectors of all samples in the training set. The similarity metric used is generally a distance measure such as the cosine distance function given in equation 2.2.

$$SIM(V_j, X) = \frac{\sum_{k \in (V_j \bigcap X)} (w_{k,j} \times f_k)}{\sqrt{\sum_{k=1}^{n} (w_{k,j})^2 \cdot \sum_{k=1}^{n} (f_k)^2}} \tag{2.2}$$

where, V_j is the vector corresponding to the j^{th} document in the training set
X is the unknown sample to be classified
$w_{k,j}$ is the weight of the k^{th} term in document j
f_k is the weight of the same term in the test sample and
the denominator is the norm of the two document vectors

Only those terms that occur both in the test and training documents are considered. This similarity measure produces a high value when the two vectors being compared are similar. A value of 1 indicates the two vectors are identical, while a similarity value of 0 indicates that the two are unrelated.

The training examples are ranked according to their distance from the test sample and the k nearest examples are selected. To assign a class label to the test sample, weighting schemes can be devised, but a simple rule that assigns a class label that corresponds to the majority of its neighbors can be used. The performance of k-NN again, is among the best for techniques proposed for text classification [34]. The classifier performs well

as it uses the majority decision of k training samples. Due to the same, the effect of noisy data is also reduced. The drawback of the approach is the presence of a large feature space, which can become problematic as the size of the training set increases.

2.1.3 Naive Bayesian Classifier

The naive Bayesian classifier is a probability based classifier that assigns class membership or a posterior probability value to a sample based on a combination of the prior probability of occurrence of a class and probabilities of the terms (also called the likelihood), given they belong to that class. For the text classification task, this translates to the combination of the probabilities of terms and the existing categories to predict the category of a given document. Using the Bayes rule we can predict the posterior probability of a category C_j among a set of possible categories $C = C_1, C_2, C_3.....C_n$ and given a set of terms $T = t_1, t_2, t_3.....t_n$ as

$$p\left(C_j|t_1, t_2, t_3.....t_n\right) \propto p\left(t_1, t_2, t_3.....t_n|C_j\right)p\left(C_j\right) \tag{2.3}$$

Naive Bayesian classifiers make a simplifying assumption of term independence (albeit incorrectly), that the conditional probability of a term occurring in a document is independent of the conditional probabilities of other terms that appear in that document i.e.,

$$p\left(T|C_j\right) \propto \sum_{k=1}^{n}(t_k|C_j) \tag{2.4}$$

Using this naive assumption, the posterior probability can be re-written as

$$p\left(C_j|T\right) \ \propto p\left(C_j\right)\sum_{k=1}^{n}(t_k|C_j) \tag{2.5}$$

Although the assumption made is strong and not often accurate, it does simplify the computation of term probabilities (as they can be calculated independent of each other). The performance of the classifier is good and compares well with other sophisticated tech-

niques such as decision trees and neural networks. In our evaluation, we have compared the performance of the $InfoSift$ system with the probabilistic naive Bayesian classifier.

From the above discussions on classification techniques, it is clear that the problem has been studied in depth and different methodologies have been applied to solve the task at hand. With this discussion on text classification techniques, we now present the relevant work in the area of email classification.

2.2 Work on Email Classification

As we have stated before, the problem of email classification presents certain features that are not found in text classification. Certain characteristics of the domain (e.g., information contained in the headers and so on) have to be taken into account to ensure good classification. Many text classification techniques have been applied to the problem of email classification. Based on the mechanism used, email classification schemes can be broadly categorized into: i) Rule based classification, ii) Information Retrieval based classification and iii) Machine Learning based classification techniques. In the sections that follow, we present an outline of some systems that have been developed to automate the task of email classification.

2.2.1 Rule Based Classification

Rule based classification systems use rules to classify emails into folders. William Cohen [22] uses the *RIPPER* learning algorithm to induce "keyword spotting rules" for email classification. RIPPER is a propositional learner capable of handling large data sets [35]. Cohen argues that keyword spotting is more useful as it induces an understandable description of the email filter. The *RIPPER* system is compared with a traditional IR method based on the *TF-IDF* weighting scheme and both show similar accuracy. The *i-ems* (Intelligent Mail Sorter) [36] rule based classification system learns

rules based only on sender information and keywords. *Ishmail* [37] is another rule-based classifier integrated with the Emacs mail program Rmail.

Although rules are easy for people to understand [38], managing a rule set may not be so. As the number and characteristics of incoming emails change, the rules in the rule set may have to be modified to reflect the same. This puts a cognitive burden on the user to review and update the rule-set from time to time, often involving a complete re-writing of rules.

Most of the email managers (e.g., outlook, eudora), allow users' to set rules for classifying email to folders. These rules have to be specified manually and can use words from various categories. The main problem here is in the manual specification and management of these rules which can become cumbersome and need to be changed often to make them work properly.

2.2.2 Information Retrieval Based Classification

Segal and Kephart [24] use the TF-IDF classifier as the means for classification in *SwiftFile*, which is implemented as an add-on to Lotus Notes. The system predicts three likely destination folders for every incoming email message. The TF-IDF classifier is based on the TF-IDF technique used in information retrieval. For each email folder, a vector of terms that are frequent across the emails in the folder (term frequency) and infrequent across other folders (inverse document frequency) is created. The set of terms thus selected is capable of discriminating the features of a given folder with those of other folders. To classify an incoming email, the term frequency vector of the email is constructed. It is compared with the TF-IDF vectors of all folders using a cosine similarity metric that is similar to the one stated in equation 2.2. The new email is classified to the folder where the value of the cosine distance function is maximum.

The TF-IDF classifier performs well even in the absence of large training data and the classifier accuracy remains reasonable as the amount of training data increases, adding to the heterogeneity of a folder. The classifier learns incrementally with every new message that is added or deleted from a folder, eliminating the need for re-training from scratch.

2.2.3 Machine Learning Based Classification

Various machine learning based classification systems have been developed. The *iFile* system by Rennie [23] uses the naive Bayes approach for effective training, providing good classification accuracy, and for performing iterative learning. The naive Bayesian probabilistic classifier has also been used to filter junk email effectively as shown by Sahami et.al [39]. The *Re:Agent* email classifier by Boone [21] first uses the TF-IDF measure to extract useful features from the mails and then predicts the actions to be performed using the trained data and a set of keywords. It uses the nearest neighbor classifier and a neural network for prediction purposes and compares the results obtained with the standard IR, TF-IDF algorithm. Mail Agent Interface (*Magi*) by Payne and Edwards [40] uses the symbolic rule induction system *CN2* [41] to induce a user-profile from observations of user interactions. The system suggests actions such as 'delete', 'forward' and so on for each new email message based on the training, hence results for multi-class categorization are difficult to assess.

From the discussion on email classification techniques, it is clear that a classifier should be able to learn from the email environment of the user. The learnt information should be used to automatically file emails to the corresponding folders or to provide intelligent suggestions to the user. The information contained within the email headers is important as many systems that learn rules based on the same or derive features from the information in the headers consider it useful for classification. The use of domain

knowledge for classification when available, adds to the set of features to make a domain informed decision during classification. We will now move onto the problem of web page classification, which again, can make use of the features that are unique to the domain of the web.

2.3 Web Page Classification

We will now consider in brief the relevant work in the area of web page classification. Although it may seem that text classification techniques can be applied to solve the problem of web page classification, it may not be as straightforward since HTML pages have an underlying structure represented by the various tag elements. It is important to take into account this structural information for classification. Attardi, Gulli et.al., [42] argue that conventional text classification techniques are content driven and to not exploit the hypertext nature of the web that is characterized by linked pages that have structure. They believe that web page classification needs to be context driven and must derive useful information from the structure and link information. The idea that the content in a link and the context around it must provide enough information to classify the document pointed by the link is the main motivation. The authors claim that a blurb of a page provides significant information about the content of page in a concise manner than the page itself (thereby reducing the noise). The information contained within the links that point to a given page and the context around them are used to assign a category to the page.

Schenker, Last et.al., [43] have used graph models for classifying web documents. The graph is constructed from the text of the web page and the words contained in the title and hyperlinks. An extension of the k-NN algorithm is used to handle graph based data. The graph theoretical-distance measure for computing the distance translates to the maximal common subgraph distance proposed in [44]. The graph model for classifying

web pages is compared with the k-NN algorithm that uses the conventional feature vector approach. The performance of the graph model is better than the conventional bag-of-words approach and is also more efficient.

The approach proposed in this thesis is different from the earlier approaches applied to the problem of document classification. It is also different from the approaches that have been attempted for email classification. Though a graph based model has been used for classifying web pages, a conventional classification technique adapted to work with graphs is used for the actual classification. To the best of our knowledge, we are not aware of any work on the use of graph mining techniques for text, email or web page classification. With this overview of the related literature in the area of document classification, the discussion on graph mining and graph mining techniques is presented in the next chapter.

CHAPTER 3

GRAPH MINING

Graph mining, as opposed to transaction mining (association rules, decision trees and others) is suitable for mining structural data. Graph mining is certainly appropriate for mining on chemical compounds, proteins, DNA etc., that have an inherent structure. The complex relationships that exist between entities that comprise these structures (for example, the double and triple bonds between carbon and other elements in a complex organic compound, the helical structure of DNA molecules and so on) can be faithfully represented using graph format.

A graph representation comes across as a natural choice for representing complex relationships because it is relatively simple to visualize as compared to a transactional representation. The various associations between objects in a complex structure are easy to understand and represent graphically. Most importantly, the representation in graph format preserves the structural information of the original application which may be lost when the problem is translated/mapped to other representation schemes. Consider the example shown in the figure 3.1[1], which consists of objects of different shapes resting on a table. Representing the same scene in graph format enables quick visualization and identification of the recurring pattern of objects and their specific placements.

3.1 Overview

Graph mining aims at discovering interesting and repetitive patterns/substructures within structural data. Relevant work in graph mining includes the Subdue substructure

[1]Courtesy: Graph Based Data Mining, Cook and Holder

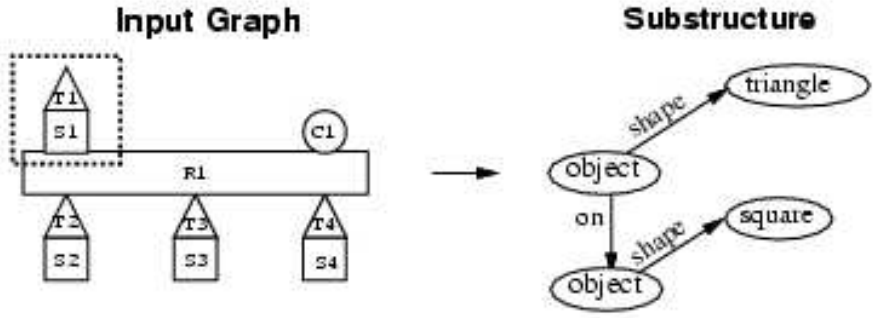

Figure 3.1 Example Substructure in Graph Form

discovery system by Cook and Holder [27]. Subdue discovers interesting and repetitive substructures in graph representations of data. It employs beam search to discover interesting subgraphs and compresses the original graph with instances of the discovered substructures. The frequent subgraphs approach by Kuramochi and Karypis [28] maps the Apriori [26] algorithm to structural data represented in the form of a labelled graph and finds frequent itemsets that correspond to recurring subgraphs in the input. gSpan [30] uses a canonical representation by mapping each graph to a code and uses depth-first search to mine frequent subgraphs.

We believe that emails can be represented using structural relationships as an email message has a structure in the form of the information contained in the headers, the subject and the body. This is radically different from the existing corpora of work where emails are considered as general text having no particular structure. The same is also true of text and web documents that have a structure in the form of the title, keywords, section headings, the HTML tag elements in case of web pages and the document body. By attaching structural information to emails and text they can be made amenable to graph mining.

Our work consists of identifying patterns within documents of a class and using these patterns for classification. An unknown sample will not match a learned pattern

exactly, for that matter, neither will two training samples match exactly. It is therefore important to allow some amount of inexactness during pattern matching. Besides, a means of substructure discovery directed by specific constraints (e.g., restraining the size of discovered substructures) is also required. Any graph mining technique that satisfies the above constraints can be used as a tool for pattern discovery. FSG and gSpan, both work with a canonical labelling of sorts and hence and do not have a notion of matching inexactly within a specified threshold or bound. They return all substructures that satisfy a given support value and do not have the capability to affect substructure discovery based on specified constraints. The Subdue substructure discovery system satisfies our requirements of inexact graph match and guided discovery and hence we have chosen the same for our work. The following sections discuss certain aspects of Subdue including the heuristics employed for substructure discovery, the process of graph matching and input/output parameters of the system.

3.2 Subdue Substructure Discovery System

The Subdue substructure discovery algorithm [27] is a graph based mining algorithm that discovers repetitive and interesting substructures in graph representations of data. Within the representation, entities and objects are mapped to vertices and the relationship between objects is represented as an edge between the corresponding pair of vertices. A substructure is a connected subgraph within the graph representation. An instance of a substructure in an input graph is a set of vertices and edges from the input graph that match, graph theoretically, to the graphical representation of the substructure.

The input to the Subdue substructure discovery system is a forest of graphs and the output is the set of best subgraphs that compress the input graph using the minimum description length (MDL) principle. The input is in the form of a file with the list of unique vertices in the graph and the corresponding edges between them. The output

consists of the best substructures discovered in the input graph, where each output substructure is qualified by its size and occurrence frequency in the input graph. For the graph in figure 3.2 the input file to Subdue is as shown in 3.3.

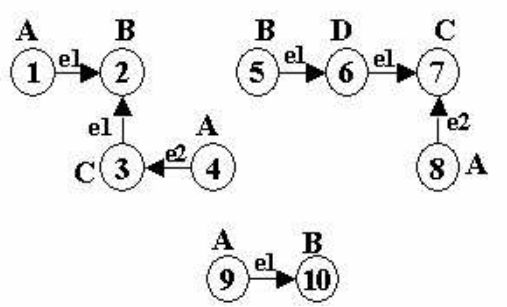

Figure 3.2 Sample Graph

```
V 1 A
V 2 B
V 3 C
V 4 A
V 5 B
V 6 D
V 7 C
V 8 A
V 9 A
V 10 B
D 1 2 e1
D 3 2 e1
D 4 3 e2
D 5 6 e1
D 7 6 e1
D 8 7 e2
D 9 10 e1
```

Figure 3.3 Subdue Input File

The following section provides a brief description of the discovery process. For a detailed explanation of the Subdue discovery algorithm, the reader is referred to [27].

3.2.1 Substructure Discovery

Subdue uses beam search for substructure discovery. It maintains three lists during the discovery process, one of which is the parent list that consists of the current set of substructures to be expanded. The substructure discovery process proceeds with the expansion of each substructure in the parent list and its evaluation based on the chosen heuristic. The discovery process halts when there are no more substructures remaining for expansion or when the user provides a terminating argument. The Subdue algorithm is presented below.

```
Subdue(Graph, BeamWidth, MaxBest, MaxSubSize, Limit)
ParentList = {} ChildList = {}
BestList = {} ProcessedSubs = 0
Create a substructure from each unique vertex label and its single-vertex instances;
insert the resulting substructures in ParentList
while ProcessedSubs <= Limit and ParentList is not empty do
    while ParentList is not empty do
        Parent = RemoveHead (ParentList)
        Extend each instance of Parent in all possible ways
        Group the extended instances into Child substructures
        for each Child do
            if SizeOf( Child ) <= MaxSubSize then
                Evaluate the Child
                Insert Child in ChildList in order by value
            if Length( ChildList ) > BeamWidth then
                Destroy the substructure at the end of ChildList
        ProcessedSubs = ProcessedSubs + 1
```

```
        Insert Parent in BestList in order by value
        if Length( BestList ) > MaxBest then
            Destroy the substructure at the end of BestList
    Switch ParentList and ChildList
    return BestList
```

3.2.1.1 Algorithm Flow

The algorithm starts with the initialization of the parent, child and best lists to the empty set. The parent list consists of the current substructures to be expanded, the child list contains the substructures that have been expanded and the best list contains the current set of best substructures. The count of the total number of substructures processed so far is maintained by the *ProcessedSubs* parameter. The unique vertex labels in the graph are inserted into the parent list. Each substructure in the parent list is expanded in all possible ways either by adding a vertex and an edge or an edge if both vertices are already present. The first instance of a new substructure becomes a new child substructure. All the other child instances that were expanded in the same way become instances of the child substructure. Each child substructure is evaluated using the MDL heuristic and inserted in the child list in the decreasing order of MDL value. The beam value is enforced on all three lists, and substructures above the beam value are removed. The parent list is swapped with the child list, which is then considered for extensions.

The input graph is compressed by replacing each instance of the best substructure by a single node. The resulting input graph is then used for the next iteration to find other interesting substructures. This process continues until the number of iterations specified by the user is reached, or the algorithm fails to find a substructure that compresses the graph. Various halting conditions for the algorithm can be specified by the user, for

instance, the *Limit* parameter limits the total number of substructures processed so far. The *prune* parameter discards all child substructures whose MDL value is lesser than its parents' and acts as a halting condition when no child substructures are left in the child list on pruning.

In many application domains, the occurrence of substructure instances that match exactly is very unlikely. It is therefore important to discover substructure instances that are similar and not necessarily exact matches. Subdue is capable of identifying both exact and inexact (or isomorphic) substructures in the input graph. It uses a branch and bound algorithm that runs in polynomial time for inexact graph match and identifies graphs that differ by vertex or edge labels using a threshold parameter that characterizes the inexactness. The aims of the discovery algorithm are two-fold, to discover repetitive and interesting substructures or patterns and to compress the original graph by abstracting instances of these substructures to provide a hierarchical or layered view of the original data in terms of the discovered substructures. The heuristic employed by Subdue for substructure discovery is based on the ability of a substructure to compress the input graph and is briefly discussed below.

3.2.2 Compression

Subdue uses two schemes for compression, the first based on the MDL principle and the second based on size. The MDL principle has been used in various applications such as decision tree induction, image processing and others. The principle described by Rinssanen [45], states that the best theory to describe a set of data is one that minimizes the description length of the entire data set. Subdue uses MDL to determine the best substructure. The best substructure is one that minimizes the description length of the original input graph. Accordingly, the description length of the input graph is then $DL(S) + DL(G|S)$, where S is the discovered substructure, G is the input graph, $DL(S)$

is the number of bits required to encode the substructure, and $DL(G|S)$ is the number of bits required to encode the input graph G after it has been compressed using substructure S. If $DL(G)$ represents the number of bits required to encode the input graph G, the final MDL value of the graph is defined as

$$MDL = \frac{DL(G)}{DL(S) + DL(G|S)} \tag{3.1}$$

A higher MDL value signifies a substructure that reduces the description length of the original data or in other words, compresses it better. The compression value is defined as

$$Compression = \frac{1}{MDL} \tag{3.2}$$

The second compression scheme, based only on size, uses a simple and more efficient but less accurate measure as compared to the MDL metric. The value of a substructure S in graph G is

$$\frac{Size(G)}{(Size(S) + Size(G|S))} \tag{3.3}$$

Here, *Size (G) = Number of vertices (G) + Number of edges (G)*

Size (S) = Number of vertices (S) + Number of edges (S)

*Size(G|S) = (Number of vertices (G) - i * Number of vertices (S) + i) +*

*(Number of edges(G) - i * Number of edges(S))*

where, G is the input graph,

S is the discovered substructure,

$G|S$ is the input graph after it has been compressed by the substructure and

i is the number of substructure instances.

3.2.3 Inexact graph Match

Though exact graph match comparison discovers interesting substructures, most of the substructures in the graph may be slight variations of another substructure. Inexact

graph match allows us to combine multiple structures into a single structure both for representation and identification. In order to detect substructures that match inexactly or vary slightly in their edge or vertex descriptions, the algorithm developed by Bunke and Allerman [46] is used where each distortion is assigned a cost. A distortion is defined as the addition, deletion or substitution of vertices or edges. The two graphs are said to be isomorphic as long as the cost difference falls within the user specified threshold. In general, it is an exponential algorithm as it compares each vertex with every other vertex in the graph. However, Subdue uses a branch and bound approach that executes in polynomial time by considering a reduced number of mappings.

3.3 Parameters for Control Flow

The input to the discovery process is taken from the file as shown in fig 3.3 and the graph is constructed using these values. A number of parameters control the working of the algorithm. They are briefly described below:

1. **Limit**: This parameter specifies the number of different substructures to be considered in each iteration. The default value is *(number of vertices + number of edges)/2*.
2. **Iterations**: This parameter specifies the number of iterations to be made over the input graph. The best substructure from the previous iteration is taken to compress the graph for the next iteration. The default value is one, i.e., no compression.
3. **Threshold**: This is a parameter that provides a similarity measure for the inexact graph match. Threshold specifies how different one instance of a substructure can be from the other instance. The instances match if $matchcost(sub, inst) <= size(inst) * threshold$. The default value is 0.0, which means that the graphs match exactly. A very large value of threshold may not be meaningful as it will

match two dissimilar graphs. Currently, Subdue supports threshold values up to 0.3.

4. **Nsubs**: This argument specifies the maximum number of the best substructures returned.
5. **Prune**: If this parameter is specified, Subdue will discard child substructures that have value lesser than their parent substructure. This reduces the search space to a great extent, though certain good substructures can be missed as the evaluation heuristics are not monotonic. The default is no pruning.
6. **Beam**: Beam specifies the maximum number of substructures that are retained for expansion in each iteration of the discovery algorithm. The default value is 4.
7. **Overlap**: This parameter guides the algorithm to consider overlap in the instances of the substructures. Two or more instances of a substructure are said to overlap if they have a common substructure. Overlap plays a significant role in calculating the compression value because with overlap we have to maintain extra information. During graph compression an OVERLAP_ <iteration> edge is added between each pair of overlapping instances, and external edges to shared vertices are duplicated to all instances sharing the vertex.
8. **Size**: This parameter is used to limit the size of the substructures that are considered. Size refers to the number of vertices in the substructure. A minimum and maximum value is specified that determines the range of the size parameter.
9. **Output**: This parameter controls the screen output of Subdue. The various values are

    1 Print the best substructure found in each iteration.

    2 Prints the best n substructures, where n is the number specified in the nsubs parameter.

    3 Print the best n substructures, as well as the substructure instances.

4 Print the best n substructures along with their instances and intermediate substructures as they are discovered.

5 Same as above, prints also each substructure considered.

With this overview of Graph Mining and an introduction to the Subdue discovery system, the adaption of graph mining techniques for document classification in the *InfoSift* system is considered in the following chapter.

CHAPTER 4

INFOSIFT: SYSTEM OVERVIEW

The *InfoSift* system, developed as part of this thesis, aims at automating the process of document classification as much as possible by using graph mining techniques to classify documents into pre-defined categories. It uses the supervised approach to classification, wherein pre-classified documents (emails, text, etc.) with appropriate class labels (folder or topic/category in our case) are used for training. The overall flow of control is shown in Fig. 4.1.

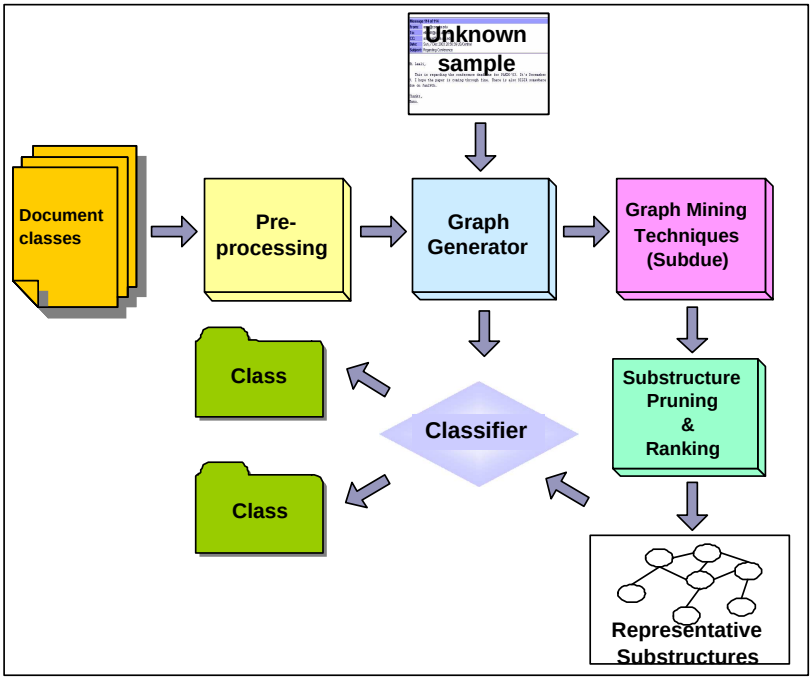

Figure 4.1 InfoSift Document Classification System

4.1 Approach Overview

The pre-classified documents in a class are used as training examples and are processed prior to graph generation. Graph mining techniques are then applied on the generated graph for pattern discovery. The resultant interesting and repetitive patterns are further processed and pruned to identify the representative set of substructures for the class. The representative substructures, once generated, are ranked for use in classification later. The classifier computes the similarity measure of the unknown sample and resolves the destination class in case of multiple matches. A brief description of the classification steps is given below.

1. **Pre-processing:** The documents in a given class form the training set for generating interesting and repetitive patterns once the stop words have been eliminated. The various characteristics of the class are taken into account to derive the optimal set of parameters for substructure discovery. Average size of documents in a class, the number of words retained from each document and the size of the document class are examples of class characteristics.
2. **Graph Representation:** The next step is to choose a graph representation that is appropriate for the domain and convert the documents in a class into the chosen graph format. Emails have a definite common structure in terms of their content (e.g., from, to, subject, body). So also do web pages in terms of the various tags elements, hyperlinks and the page text. We have proposed a canonical graph representation scheme for text, email and web pages. Alternate representation schemes that incorporate domain characteristics for emails and web pages have also been proposed. The structural nature of the web has been considered in other approaches for web page classification as it is a very obvious characteristic of the domain. However, structural representation schemes for emails, which definitely have a structure in the form of email headers and body have not been considered

before. Text also has a structure, it is not marked, but does comes across subtly in the form of the text title, section headings, the actual textual content and so on. The use of structural representation schemes for text and email classification is therefore a first.

3. **Substructure Extraction:** The Subdue substructure discovery algorithm is used to discover/extract the representative substructures from the documents represented in graph format. The substructure discovery process is driven by the parameters derived as part of the pre-processing step and are input to the Subdue discovery algorithm. Among these parameters is the *threshold* parameter that allows for the desired level of inexact graph match.
4. **Representative Substructure Pruning:** The output of the discovery process is likely to contain a large number of substructures and all of them are not likely to contribute towards classification. Furthermore, the cost of the approach will increase with the increase in the number of substructures retained and compared during classification. Since Subdue was not developed for this purpose, we need to process and prune its output to determine what is relevant for classification. Hence, our goal is to identify the subset that is needed for discriminating the unknown sample during classification. The set of substructures output by the discovery process is pruned based on a number of parameters to determine the representative set of substructures needed for the given class. Those that are retained must cover a significant portion of the class contents and have sizes that measure relatively well to the contents of the class.
5. **Representative Substructure Ranking:** Some discovered substructures are more important than others either because they appear more frequently or they measure well with respect to the sizes of a class' contents. It is therefore important to discriminate among the set of pruned representatives from the viewpoint of

classification. Each representative substructure needs to be ranked in some way to indicate its representativeness. The rank associated with a substructure is used for classifying the unknown samples. Certainly, there is a difference in a match with a highly ranked substructure versus an average one. This is especially important when the samples are classified into multiple classes.

6. **Processing to-be-classified Document:** Pre-processing (e.g., stop word removal) that is performed on the training set is also performed on the unknown sample to be classified to bring it into a canonical representation. The test sample is then converted into a graph for the purpose of classification.
7. **Classification:** The classifier compares the representative substructures with the unknown test sample (again using Subdue) to determine if any of them appear in the unknown sample graph. If a match is found, the test sample is classified to the corresponding category. For multi-way classification, in case of more than one match, the ranks associated with the matched substructures are used for resolving the destination class to which the unknown sample should be classified. It is classified to the category with the highest ranked substructure match.

4.2 Details of Document Classification

A brief overview of the control flow of the *InfoSift* text classification system was provided above. In the discussion that follows we will elaborate on the pre-processing tasks and the various graph representation schemes proposed for the different domains addressed by this thesis.

4.2.1 Document Pre-processing

Using the entire document would be an overkill for any classification mechanism. Typically, a document consists of a large number of words, not all of which are necessary

or even useful for characterizing the document. For example, articles, conjunctions, and even common words that occur frequently in a document can be pruned without affecting the outcome. Stemming is another commonly used technique to prune multiple occurrences of the stem in different forms. It is important that the original document is pre-processed appropriately as it will otherwise add noise in the form of irrelevant words and reduce the effectiveness of any approach.

Information retrieval uses several techniques for pre-processing documents to prune the size of the input without affecting the final outcome. For our purposes, we need to determine what can be pruned and what need to be retained. Before generating the input graph for the contents of a class, the documents in the class are processed to eliminate stop words. Since the goal is to retain substructures that are frequent across documents of a class, the terms that comprise the substructures have to be frequent as well. These terms must occur frequently, preferably across all the documents in the given class and not merely in a single document. This choice of retaining words that are common across documents takes care of the disparity in length, as some documents are considerably longer than others. Words are ranked based on their occurrence frequencies across all documents in a class and those whose frequencies account for more than $f\%$ of the sum of all frequencies are retained. The rationale behind this is to study the effect of f on classification. Of course, by choosing f to be 100, one would retain all the words. But, the lower frequency words may not contribute towards classification. The parameter f is tuned to observe its effect and identify any possible dependency on other class characteristics. To construct the graph, only those words in the message body that are a member of this frequent set are used. This ensures the words chosen are frequent not only in a single document, but across a substantial number of documents in the class under consideration.

An example will make this clearer. Consider the sample document (belonging to a class) shown in Fig. 4.2.

TEXAS COMMERCE BANCSHARES FILES PLAN

Texas Commerce Bancshares Inc's Texas Commerce Bank-Houston said it filed an application with the Comptroller of the Currency in an effort to create the largest banking network in Harris County.

The bank said the network would link 31 banks having 13.5 billion dollars in assets and 7.5 billion dollars in deposits.

Figure 4.2 Document Sample

To construct the graph corresponding to this document once the stop words are eliminated, the set of frequent terms across all documents is considered. Assuming the set of frequent terms is as shown in Fig. 4.3, the graph corresponding to the document consists of terms shown in Fig. 4.4

Assets
Bank
Billion
Commerce
Comptroller
Currency
Debentures
Deficit
Deposits
Dollars
Economy
Financial
Markets
Portfolios
Stocks
Trading
Venture
Wall Street
.
.

Figure 4.3 Global Set of Frequent Terms

This technique of processing important terms within documents of a class can be compared with the term frequency (TF) technique used in information retrieval. TF

Figure 4.4 Document Term Set

classifiers extract frequent terms from documents of a class to create a class descriptor, typically a term frequency vector, which is used to compare the similarity of an unknown sample with the class contents. The pre-processing that is carried out is similar to the TF classifiers in extracting frequent terms, but unlike the TF technique, a single graph of frequent terms (vis-a-vis the document vector) is not constructed. Graphs representing each document in the class are constructed from those terms in the document that appear in the frequent set.

This forest of graphs forms the input for substructure discovery. They are input as a sequence of vertex and edge descriptions. For our experiments, we have used a 80-20 and 60-40 split, i.e., either 80 or 60 percent of the documents are *randomly* chosen from a class as the training set and 20 or 40 percent are used for classification, respectively.

4.2.2 Graph Representation

The ability to capture relationships between words (or sets of words) within a document differentiates this approach from other traditional approaches where such relationships were not (or could not be) be captured. In the traditional information retrieval approaches, each word occurrence is considered independent of each other as they cannot be grouped in any way. The ability to impose structural relationships can be beneficially exploited depending upon the domain and what relationships are important to represent in a document. For example, the structure used for an email may be very different from

a structure used for a news paragraph. The ability to impose a meaningful structure allows us to make it either hierarchical, flat, or a combination thereof and provide different types of emphasis and grouping.

We believe that representations chosen appropriately can cover the range of independent word occurrences to relationships among document/email objects and as a consequence provide the context or semantics for extracting representative substructures and use them for classification. Below, we present a couple of representations and describe their characteristics.

The first step in discovering representative patterns is to transform the documents into a graph format for mining. Also, based on the domain knowledge, it is possible to choose a representation that provides different emphasis for different domains. We have proposed a graph representation scheme that can be used across the text, email and web domains. This canonical representation is shown in Fig. 4.5. It is a star-graph representation and consists of a central anchor or root vertex. The chosen words from the document form the remaining vertices, along with edges that connect them to the central root vertex with a descriptive edge label such as *contains*.

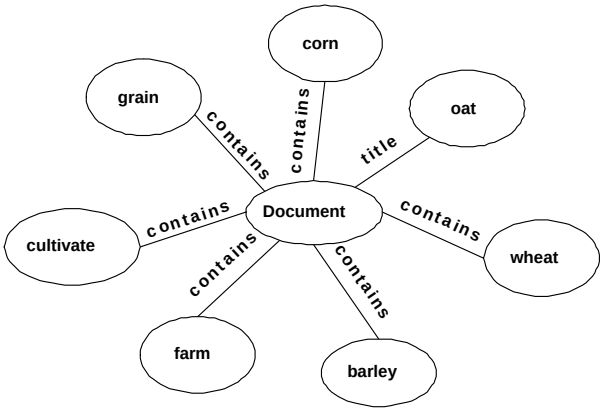

Figure 4.5 Canonical Graph Representation

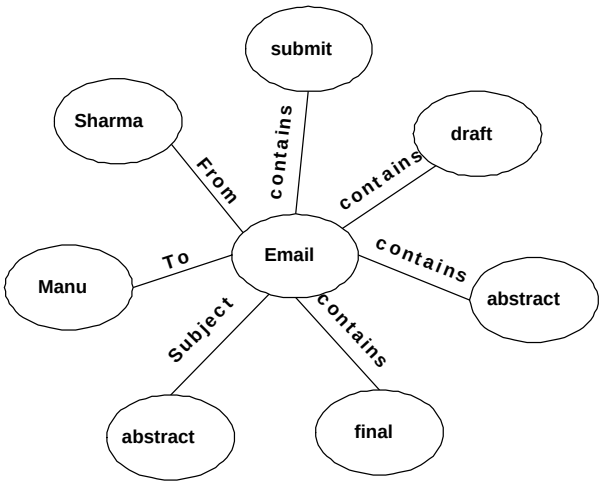

Figure 4.6 Emails Represented in the Canonical Graph Representation

The ability to label edges makes this simple representation quite effective if the labels correspond to the various components of a document, email or web page. Figure 4.6 shows such a representation where the edge labels used are $From$, $Subject$, To, and *contains* to differentiate between various significant components of an email. For email classification, this scheme considers all the information in an email message, with each word in the email connected to the central root vertex.

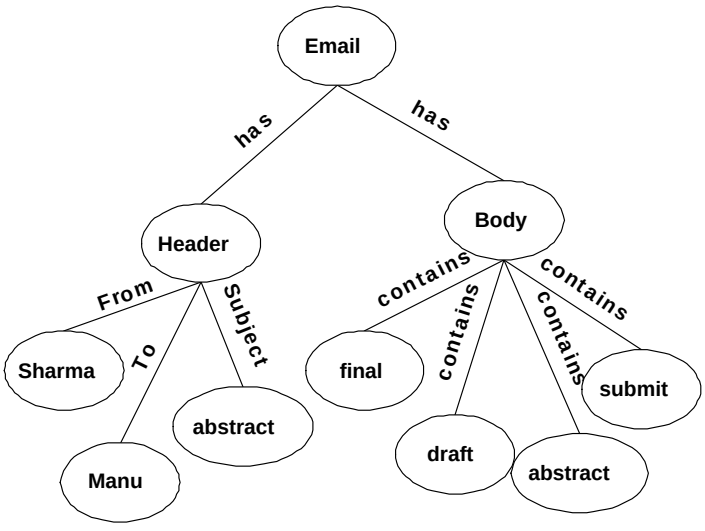

Figure 4.7 Graph Representation for the Email Domain

Of course, other representations that use the components of an email explicitly can be formulated. For example, if we want to give more emphasis to the header of an email, then we can choose a representation to reflect that. The scheme in Fig. 4.7 corresponds to the scheme specifically customized for representing emails as a graph. The representation differentiates between the email headers and the contents of the email body. It was devised to study classification using either headers only or all the information present in the email message.

Web pages can also be represented in graph form. Shown in Fig. 4.8 is the representation that takes into account the information represented by title of the page, the hyperlinks that point to other pages and the information content represented by the page text. We have considered hyperlinks, because they point to information sources relevant to the current page.

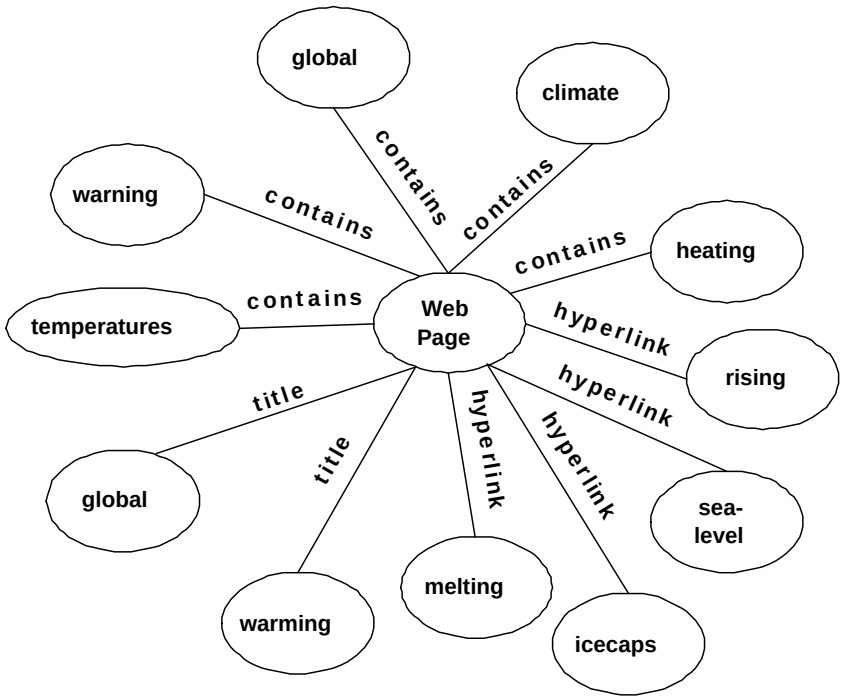

Figure 4.8 Graph Representation for the Web Domain

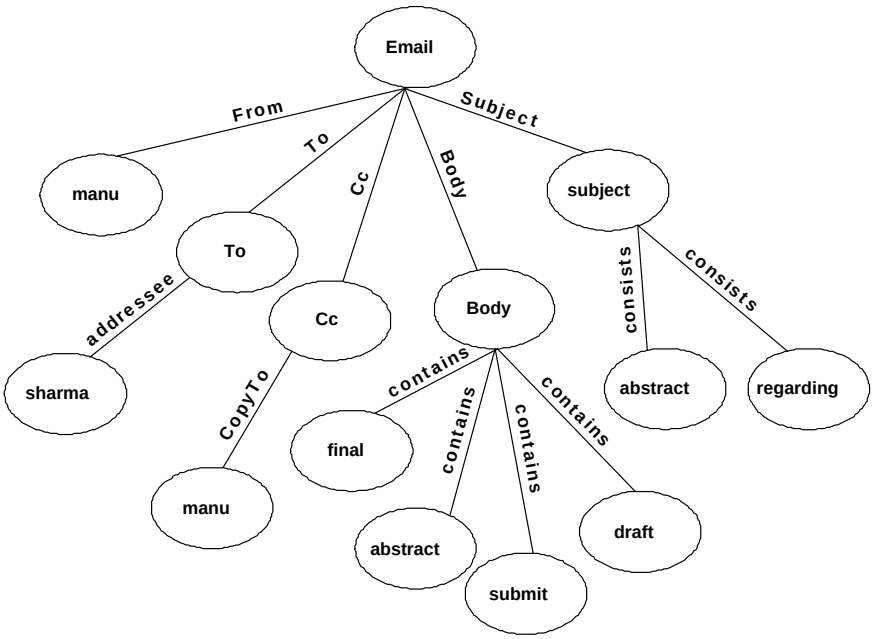

Figure 4.9 Initial Representation for the Email Domain

During the process of determining the representation schemes that would capture enough and appropriate information for classification, we also considered a number of other alternatives, which led us to the schemes which we have eventually used to establish the effectiveness of our approach. One of the representation schemes considered is shown in Fig. 4.9. It shows each component of the email, except for the 'From' header at finer levels of granularity. In the scheme eventually used for representation, the email is divided into 'Header' and 'Body' components only, with the various header elements grouped under a common 'Header' tag. The granularity at which we want to model the input can be effectively controlled using the graph representation. A salient feature of our approach is that the rest of the techniques used is independent of the representation. In other words, the representation changes the type of representative substructures generated and the overall approach remains the same.

Once the documents have been represented in graph format, they can be mined for representatives. The graphs corresponding to the documents in a class are input as a file to the Subdue system. The input file consists of vertex and edge entries. Each vertex entry associates a unique vertex id with every vertex label. Each entry corresponding to an edge represents an undirected edge between a pair of vertices and the associated edge label. The input file to the Subdue system corresponding to the representation in Fig. 4.6 is shown in figure 4.10.

```
v 1 Root
v 2 sharma@cse.uta.edu
v 3 manu@cse.uta.edu
v 4 abstract
v 5 final
v 6 draft
v 7 abstract
v 8 submit
u 1 2 From
u 1 3 To
u 1 4 Subject
u 1 5 contains
u 1 6 contains
u 1 7 contains
u 1 8 contains
```

Figure 4.10 Input Graph File

The discovery process is driven by certain parameters that are determined as a part of the pre-processing and graph generation phase. As the discussion of parameters warrants a detailed study they are covered fully in the next chapter.

CHAPTER 5

ESTABLISHING THE FOUNDATION OF INFOSIFT

The goal of the $InfoSift$ system is to identify representative structures of a given class of documents and use the same for classification. In order to achieve this, we have to choose a number of input parameters for the Subdue algorithm to effect substructure discovery (such as *beam*, *threshold*, etc.,). We believe that the class itself needs to be used as a source for deriving these parameters as classes vary in their characteristics. This way, the representative structures generated will be customized to a class and not based on a fixed set of parameter values. To determine the representative structures that best characterize a particular class, certain characteristics of the class need to be derived. We need to identify the parameters that are important and then provide a way for deriving them with substantiation. These parameters must be tunable and effective for diverse document and class characteristics.

5.1 Impact of Class Characteristics

Not all classes exhibit similar properties; certain classes may be more dense as compared to others and certain others may have larger document content providing a greater amount of information for training the classifier. Also, in case of an email hierarchy, emails in a particular folder may exhibit greater cohesion than those in other folders, making it easier to determine the similarities among them. Due to constant addition, deletion and movement of emails across folders, the folder contents change with time. Hence, these and other class characteristics need to be quantified and specified as input parameters to the Subdue discovery algorithm to ensure that the substructure

discovery process is based on the traits of the class. If the discovery process is guided by these parameters, the representative substructures generated are likely to better reflect the contents of the class. Below, we discuss the key characteristics of classes that we believe affect substructure discovery and how they can be mapped as parameters for the same. These parameters have been used for performing extensive experiments to validate our premise.

5.1.1 Average Document Size and Threshold

In the textual domain, it is hard to find instances that match exactly. For the purpose of classification, it is important to be able to allow some flexibility in matching instances that are alike and not merely exact. This leeway should be applicable both while building a descriptor for the document class and while comparing an unknown sample with the class contents. The matching of similar instances is carried out by the process of inexact graph match in our work. With inexact graph match, instances of substructures that differ slightly in their vertex or edge descriptions are considered alike. Inexact graph matching is used for both pattern/substructure discovery and classification.

As discussed earlier, the *threshold* parameter determines the amount of inexactness allowed during substructure discovery. During inexact graph match, *threshold* determines the number of vertices and edges that vary among instances of the same substructure. The actual number is determined by

$$(num\,of\,vertices + num\,of\,edges) \times threshold \tag{5.1}$$

Since each document in a class has a different size (even after pre-processing), the average size of documents in a class is used as one of the characteristics of a class. We argue that a low value of *threshold* allows for a significant amount of inexactness while comparing substructure instances of documents that contain a large number of words. This is

because, for a large document, even with a low *threshold*, the actual value as determined by equation 5.1 will allow a reasonable amount of inexactness. This ensures that similar substructures with slight variations are identified. For documents with relatively smaller content and hence fewer vertices in the input graph representation, a larger value of threshold is required for inexact match that allows the same proportion of variations as in the former case. Using the size of the documents in a class, we can always determine the amount of inexactness to allow for graph match. If the amount of inexactness to be allowed in terms of the number of edge/vertex label variations is 'i', then the particular value of threshold is obtained by

$$threshold = \frac{i}{avg_s} \tag{5.2}$$

where, avg_s is the average size of the documents in the class

For example, say we wish to limit the variation in the vertex and edge descriptions to a maximum of six differences ensuring that the matches preserve the bounds on similarity. The above formula will then derive the right value of threshold taking into account the size of the documents in the class. For smaller sized documents, the value of threshold will be large as compared to that for larger sized documents. In any case, the number of variations between instances will be capped at most at six.

Here, we have interpreted the average document size as a parameter that affects pattern discovery and used it to derive the right value of threshold that allows for a reasonable amount of variation and at the same time preserves the similarity between substructure instances. It is also important to make sure that the value of i is not very large allowing very dissimilar substructure instances to be grouped as identical. A value of i more than 10 may not be appropriate even for a class with a large value of average document size.

5.1.2 Document & Class Size Vs Number of Substructures

The number of structures returned by Subdue as the set of best substructures is limited by the parameter *nsubs*. To ensure that the representative set consists of substructures that characterize the class, the number of substructures to be returned by the discovery process has to be derived from the class characteristics. If the average size of the documents in a class is large, then the number of subgraphs is also large. Similarly, if there are a large number of documents in a class, there probably will be a large number of substructure instances as well as substructures themselves, that characterize the class. We have derived the number of substructures by using both the class size and the average document size along with weights to emphasize each factor. The formula for the same is given in the following equation

$$nsubs = w_1 \times C_s + w_2 \times avg_s,\ w_1 > w_2 \tag{5.3}$$

where, C_s is the size of the class and

w_1 is the weighting factor applied to the same

avg_s is the average size of the documents in the class and

w_2 is the weight applied to the average document size

As evident, the effect of class size is pronounced with increased weight assignment for the same. The particular numeric value for w_1 depends upon the size of the class under consideration. Classes have been discriminated into small (less 60 documents), medium (61 to 200) and large (greater than 200 documents) based on their sizes. We will now discuss the rationale for determining the particular values of w_1 and w_2 depending on class characteristics.

5.1.2.1 Small-sized Classes

Since Subdue picks substructures based on their ability to compress the original graph, for a small class, with fewer documents, large substructures despite their low frequencies are picked as best substructures, because abstracting even their few instances results in greater compression of the original graph. To ensure that comparatively smaller sized repetitive substructures are picked, a large value of *nsubs* is required. Also, as the average document size increases, the probability of an increase in the number of substructures is higher. Therefore, taking into account the need for a large '*nsubs*' with a small class size and scaling it with increase in average document size, we arrive at the following weights for w_1 and w_2

$$nsubs = 1.25 \times C_s + 0.50 \times avg_s \tag{5.4}$$

where, C_s is the size of the class and

avg_s is the average size of the documents in the class

The weight w_2 is fixed at 0.50 for all average document sizes as the product term scales with increase in average document size. These values have been determined based on experimental observations. Similar weight measures specific to medium and large sized classes have also been derived by experimentation.

5.1.2.2 Medium-sized Classes

For medium-sized classes, in our case those that have about 60-199 documents, it is likely that repetitive substructures rather than isolated instances of long substructures will be reported as best substructures. For the same reason, the value of nsubs can be taken as a fraction of the class size and scaled with an increase in average document size, leading to the formula given below:

$$nsubs = 0.90 \times C_s + 0.50 \times avg_s \tag{5.5}$$

where, C_s is the size of the class and

avg_s is the average size of the documents in the class

5.1.2.3 Large-sized Classes

Classes that contain more than 200 documents are characterized as large. An increase in class size thereafter will serve to increase substructure instances rather than the number of substructures themselves. Any pattern that has to be reported as interesting in such a large class will have to have a considerable frequency of occurrence. Therefore, for such large-sized classes, the term corresponding to the same can be fixed at a constant value irrespective of class size. In the computation of 'nsubs', the term corresponding to the class size has been capped at 150, which we believe is a reasonable value. To maintain consistency with the previous cases, the average document size can be scaled as before. The equation for determining the number of substructures for large classes is shown below:

$$nsubs = 150 + 0.50 \times avg_s \tag{5.6}$$

where, C_s is the size of the class and

avg_s is the average size of the documents in the class

5.1.3 Beam Size

Beam determines the number of best substructures retained at the end of each iteration of the discovery algorithm. The discussion regarding parameter 'nsubs' entailed ensuring the value chosen for the number of substructures is optimal to cover the possible set of substructures. While this is required, it is also important to ensure that the interesting substructures discovered during each iteration of the discovery algorithm are

available for further consideration. This can be effected by selecting the right value of beam size.

The value of the *beam* chosen is in proportion to class size. Large sized classes typically contain many patterns owing to the presence of a large number of documents. For the email domain, accumulation of email in folders may lead to an increase in the heterogeneity making it to hard to identify and associate definite patterns with the folder. Hence, a relatively large value of beam is required to guarantee interesting substructure discovery. A low value of *beam* will result in the loss of some interesting substructures as they do not get a chance to be evaluated in subsequent iterations of the discovery algorithm. Experiments employing *beam* sizes of 2, 4, 6, 8, 10 and 12 were performed on the experimental classes. Depending upon class size, in almost all cases *beam* values of 12 worked well for large classes. For small and medium sized classes, beam values that returned good results were 2 and 4 respectively. A larger value of *beam* for these classes only leads to increased computation time and extra processing in terms of pruning the unnecessary substructures.

5.1.4 Substructure Size Vs. Minsize

Consider the case of classifying emails into folders. Substructures that are common across all emails (and across all folders) do not contribute to the classification process as they provide no discriminating capability. For instance, a substructure that contains information regarding only the sender and addressee of an email will not help in classification as emails with the same information may be classified into different folders. In this case, it is important to have additional information to enable successful classification. The representative substructures chosen should provide enough information for discriminating amongst folders. The substructure size should be constrained above a minimum to pick substructures that contain something more than a common 'core'.

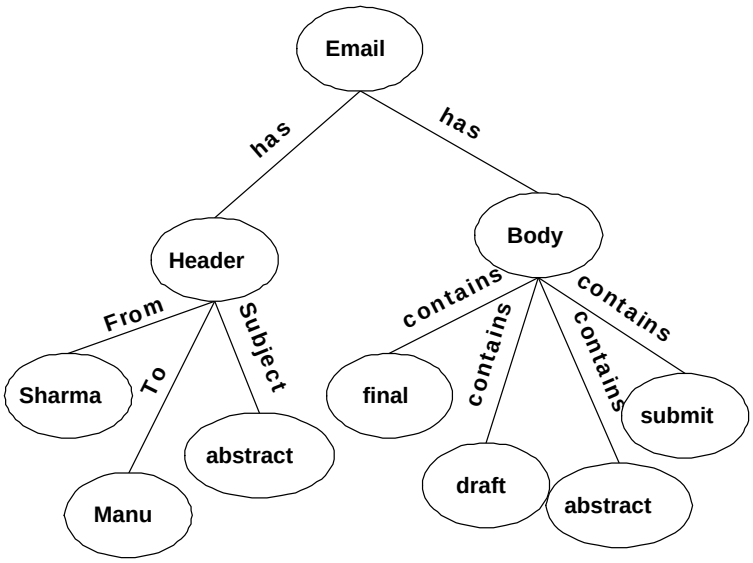

Figure 5.1 Graph Representation for the Email Domain

From the graph representations of Fig 5.1 it can be inferred that the smallest sized substructure will contain at least four vertices. These are the Email, Header, and at least two among the 'To', 'From', 'Sender', 'Cc' address fields and the 'Subject' field if all the other fields are different. Substructures smaller than this are common to all emails within a folder and also across all folders. Therefore, the minimum size of the substructures to be reported is constrained to be four. In fact, using a lower minimum size may affect the classification adversely. This constraint needs to be determined from the graph representation scheme employed.

Applying the same argument, for the representation in Fig. 5.2, the minimum size of the substructure should be three. This ensures that substructures that are picked have sizes greater than the size of substructures that are most likely to be common across many email folders, and hence capable of discriminating between the same.

For the text and web domains too, it is important that the substructures chosen are not very small in size. We have constrained the minimum size of the substructures to

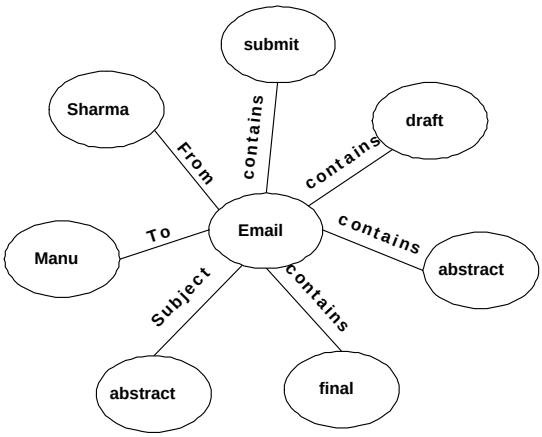

Figure 5.2 Emails Represented in the Canonical Graph Representation

be three for the reasons stated above. This constraint can be specified using the *minsize* parameter that is input to the Subdue system.

5.2 Substructure Pruning

A large number of substructures are returned as part of the discovery process, one alternative is to retain all of them. However, they are all not interesting from the viewpoint of classification. Inexact graph match returns substructures similar to others in terms of occurrence frequency and substructure size, while varying in one or more vertex or edge descriptions. Retaining several substructures that have the same frequency and size and vary only slightly in content (duplicates) will not contribute towards classification especially when inexact match is used. It is therefore necessary to prune this set of substructures to retain only those that are true representatives of a class. This translates not only to savings in space but also a reduction in computation during classification, as comparison with duplicates (as per our definition) is avoided.

As described, inexact graph match returns some substructures that are similar to others in terms of occurrence frequency and substructure size, while varying in one or more vertex or edge descriptions. To prune the possibly large set of substructures,

only those that differ in frequency and size are retained. This ensures uniqueness of the substructures. For instance, two substructures with six vertices each and different occurrence frequency are obviously different, as the same substructure will not be reported twice with different frequencies of occurrence. Therefore, each substructure in the representative set refers to a unique pattern that follows from the documents of the class under consideration.

Since the discovery algorithm uses compression as a heuristic, it also identifies certain large substructures that do not occur very frequently. This is due to the fact that abstracting such isolated, but large-sized substructures greatly compresses the original input graph. Hence, the discovery process returns such substructures as interesting, even though they do not occur repetitively. These substructures do not significantly add to the representative set primarily because they do not cover a substantial portion of the class contents. Substructures with very low frequencies as compared to the class size are therefore eliminated from consideration as potential representative substructures. The representative set of substructures generated after pruning is then ranked and used for classification.

5.3 Substructure Ranking

The representative substructures are ranked based on the frequency of occurrence, average size of documents in the class under consideration and substructure size. Substructures with sizes comparable to the average document size and which cover a reasonable number of documents are preferred, as this signifies greater correlation with class contents. The formula for rank takes into account the above and is computed using the following equation

$$R_s = \frac{S_s}{avg_s} \cdot \frac{f_s}{f_L} \tag{5.7}$$

where, S_s is the size of the substructure

avg_s is the average document size

f_s is the occurrence frequency of the given substructure and

f_L is the frequency of the most repeated substructure

The above formula, therefore assigns a higher value of rank to substructures that not only cover a significant percentage of the input set, but also compare well with the average size of the documents in the same. Relatively large sized frequent substructures signify greater similarity among documents of a class. An incoming document that matches a substructure with a high value of rank in class C_1 as compared to an average ranked substructure in class C_2, is rightly classified to C_1.

5.4 Classification

The best n substructures that are chosen as representative substructures are ranked as explained previously. Obviously, n varies with the class and average document size. These ranked representative substructures are then used for classification. The unknown sample is compared with the set of ranked representative substructures of all classes. As with the generation of representative pattern subgraphs, inexact graph match is used for comparing the unknown test sample with the predefined representative substructures. The sample document is filed to the corresponding class if a match occurs. Since, the representative substructures of each class are ranked, in case of multiple matches, the test sample is filed to the class with the highest ranked substructure match signifying higher correlation with the class contents.

We have therefore considered the parameters we believe affect substructure discovery the most and provided means to derive them with substantiation. With this discussion we now move onto some implementation aspects and present our findings in the following chapter.

CHAPTER 6

EXPERIMENTAL EVALUATION

This chapter presents results of the extensive experimental analysis performed on various document collections, email folders and web pages. The experimental results reinforce our premise that terms in a class exhibit relationships and these patterns of term occurrences can be learnt and later used for classification. The consistent performance of the classifier across different textual domains and on varying class and document characteristics establishes the applicability of our proposed approach for document classification.

As we have considered the domains of text, emails, and web pages, the performance of the classifier for each domain is presented in detail in a separate section. The experimental setup and a brief description of the data set used is also provided. Before we discuss the results of classification, a brief overview of the system implementation is presented below.

6.1 Implementation

The *InfoSift* system is implemented in Perl. Perl has been chosen as the language of choice, as it provides excellent support for text processing and manipulation. Since the discovery algorithm is implemented in C, the choice of using an interpreted language for developing the various modules does not slow down the overall performance of the system. The modules for graph generation, substructure pruning, representative set ranking and classification have been implemented in the prototype. The input to the system consists of one or more document classes along with various parameters for graph

generation and pattern discovery. The parameters are provided in the form of a configuration file, which consists of a list of options such as the split for cross validation, choice of graph representation, beam value and so on. The prototype pre-processes the classes, generates graphs, computes the various parameters and invokes the discovery algorithm for generating representative substructures. The generated output is pruned and ranked for classification. The outcome along with a number of other values generated are logged for further analysis.

In the implementation of the various modules, we have been able to optimize greatly due to the use of Perl, which is inherently geared towards string processing and extraction. The availability of pre-developed functions for many routine tasks and the ability to handle complex data structures that have been utilized in the implementation justify its use. In the discussion that ensues we will briefly describe some of the implementation aspects of the various modules and set of configuration options used.

6.1.1 Configuration Parameters

The *InfoSift* system accepts parameters for different tasks in the form of a configuration file. We have provided options for various parameters such as choice of graph representation scheme, randomized generation of training and test data sets, the option of pruning the representative set and so on. Also, values of certain other parameters that are important for substructure discovery are provided. In the case where certain parameters are absent in the configuration specification, *InfoSift* uses default values for the same. Besides the parameters specified in the configuration settings, there are various other parameters that affect substructure discovery and are derived as part of pre-processing and graph generation. Listed below is the set of configuration file options for various parameters.

1. **Document Class:** The name of the document class or email folder that is used to train the classifier.
2. **Log File:** The file name to log the results of processing and classification for the specified class. The logged information includes the values of the various parameters used for classification, the running times for each processing step, such as graph generation, substructure discovery, pruning and so on. It also includes information regarding the substructures matched during classification, along with the corresponding rank information. In case the log file name is not specified in the configuration file, a default name derived from the class name and other attributes is used.
3. **Training Test Set Split:** The document classes provide information for training the classifier. The percentage of the class sample to be used for training can be supplied to the $InfoSift$ system by specifying the split ratio in the configuration file. For our experiments, we have used training and test splits of 80:20 and 60:40 i.e., either 80% or 60% of the sample documents in a class are used to build the classifier and the remaining 20% or 40% are used for classification.
4. **Random/Sequential Generation:** This parameter specifies whether the set used for training and testing is to be generated randomly from the class or in the case of sequential generation of the test and training set, the first $n\%$ are used for training and the remaining for classification.
5. **Seed:** In case of random generation of the training and test data for classification, a seed value can be provided for the randomized generation. If left unspecified, the system supplies a seed for the generation process. This is needed for repeating the experiment with the same pseudo-random sequence.
6. **Feature Set Size:** The top $f\%$ of the features to be selected during the preprocessing of the folder contents can be specified using this parameter.

7. **Graph Representation:** We have proposed various graph representation schemes for different domains. The choice of graph representation is input to the system using the option for the scheme in the configuration file.
8. **Graph File:** The graph corresponding to the training samples in a class are output by the graph generator to the specified output file. If the parameter is left undefined in the configuration, a default file name is used.
9. **Subdue Output File:** This parameter specifies the output file that stores the results of the pattern discovery process. As in the above case, a default value is used if the parameter is not specified.
10. **Threshold:** For inexact graph match, the amount of inexactness permissible is controlled by the *threshold* parameter. This can be specified in the configuration file, else value for the *threshold* is calculated as explained by the equation 5.1 in chapter 5.
11. **Minsize:** The minimum size of the substructures output by the discovery process can be constrained above a certain value. The minsize parameter in the configuration is used for the same.
12. **Beam:** The value for *beam* is specified using this parameter. In case the parameter is left unspecified, a small beam is used for small-sized classes and a large value of beam is used for large sized classes. The exact value is derived once the class statistics are collected upon graph generation.
13. **Prune:** This parameter can be turned on or off depending upon whether the output of the discovery process needs to be pruned or used as is for classification.

The above are the parameters that can be specified to the $InfoSift$ system. We think they cover a range of issues we have addressed as part of the thesis, including the various application domains represented by the different graph formats, the various parameters for guided discovery, the size of the feature set for classification and so on.

With this overview of the configuration settings, we can move onto the details of graph generation and the implementation issues involved.

6.1.2 Graph Generation

The documents in a class form the training samples used to derive the representative substructures of a class. The graph generator that has been developed is capable of generating graphs for various text domains addressed in this thesis. For processing emails, the perl packages Mail::Internet and Mail::Address are used to extract relevant information from the headers and the body of an email message. The HTML::TokeParser package has been used for processing web pages and deriving the necessary information from the various HTML tag elements. In our case, we are interested in the information present in the title and the body of the page, along with the text contained in the hyperlinks.

We have used an associative array (or hash) structure to store the term-frequency pairs. The documents that from the training set are used to construct a global hash of term occurrences across all documents. This set of terms is pruned depending upon the percentage of the feature set that is to be retained. During the construction of the graphs for sample documents, only those terms in the document that also occur in the global hash of all frequent terms are retained. The graph is then constructed in the format chosen for representation. As part of the graph generation process, class statistics corresponding to the class size and average document size are logged for use during pattern discovery.

6.1.3 Substructure Discovery

The process of pattern discovery is carried out by the Subdue substructure discovery system. The $InfoSift$ system invokes the discovery algorithm and supplies parameters for the same. In the previous chapter we have discussed in detail the effect of various

class and document characteristics, and have derived a means of quantifying them in a way that can be used to influence the discovery of substructures. The output of the system is written to a file, which is processed to prune substructures and generate the representatives of the class under consideration.

6.1.4 Representative Pruning & Ranking

To derive the true representatives of a class, we compare substructures to eliminate those that are similar in terms of substructure size and frequency and differ only in their description of an edge or vertex. The $InfoSift$ system analyzes the output generated by Subdue to mark a substructure and discount all others that correspond to it as per our definition of similarity, until it encounters another that differs in either of the aforementioned characteristics. In addition, certain large-sized substructures that are highly infrequent are pruned as well. This is performed for large classes (which contain over 200 documents), where substructures that occur only once or twice are eliminated as they do not aid the classification process.

The pruned substructures are ranked as described by equation 5.7. The representatives are stored in separate files and information regarding the ranked representatives is dumped into a file to be used during classification. The Data::Dumper module is used to save the information in the form of a hash data structure. The representatives are ordered based on rank and there may be several representatives with the same value of rank. For each unique rank (which is the key for the hash), information regarding all the representatives (the name of the representative file) that corresponds to that rank, along with the size and frequency of the representatives is stored. During classification, the $InfoSift$ system reads the information contained within the file and the compares the test sample with the representatives in order of rank. Once a match is found, the system stops further comparison with representatives for the corresponding class. In case

of multiple matches, the ranks of the matched representatives are used to determine the destination class for the sample.

6.2 Experimental Analysis

With the important implementation details outlined, we can now look at the classification results for the various domains. All experiments have been carried out on Pentium Xeon 2.66 Ghz dual processor machines with 2GB memory. Extensive experiments on a large number of classes with diverse characteristics (i.e., different average document size, dense, sparse classes, homogeneous and heterogeneous email folders and so on) have been carried out to study the effect of the various parameters on classification. Since each domain presents issues that are unique to it, we will consider the discussion for each domain separately.

6.2.1 Text Classification

The results for text classification are presented in this section. For the experimental analysis, we have used the Reuters-21578 corpus, which is a benchmark corpus for text categorization tasks. The corpus consists of news articles from various categories, with multiple category assignments for many documents. The category distribution of the corpus is skewed with a majority of the categories containing few documents to a few containing thousands of documents. Also, there are documents in the corpus that are unlabelled which have not been considered for our experimental analysis. The resulting set of over 13000 documents corresponding to 60 topic categories has been used for training and testing purposes.

A number of experiments have been performed to determine the viability of the proposed approach and to study the effect of various class and document characteristics on classification. We have performed extensive experiments using a training and test set

split of 80:20. The metric used for evaluation is accuracy, which is defined as the ratio of the number of correct category assignments to the number of documents to be classified. The performance of the $InfoSift$ system is also compared with the probabilistic naive Bayesian classifier, implemented in the *Bow* library developed by Andrew McCallum [1]. Each of the experimental results are now considered in detail.

6.2.1.1 InfoSift Vs Naive Bayes

The results of comparison with the naive Bayesian probabilistic classifier is presented in Fig. 6.1. The classification accuracy of InfoSift is consistently better than the Bayesian classifier, with the exception of the two large sized classes in the training set.

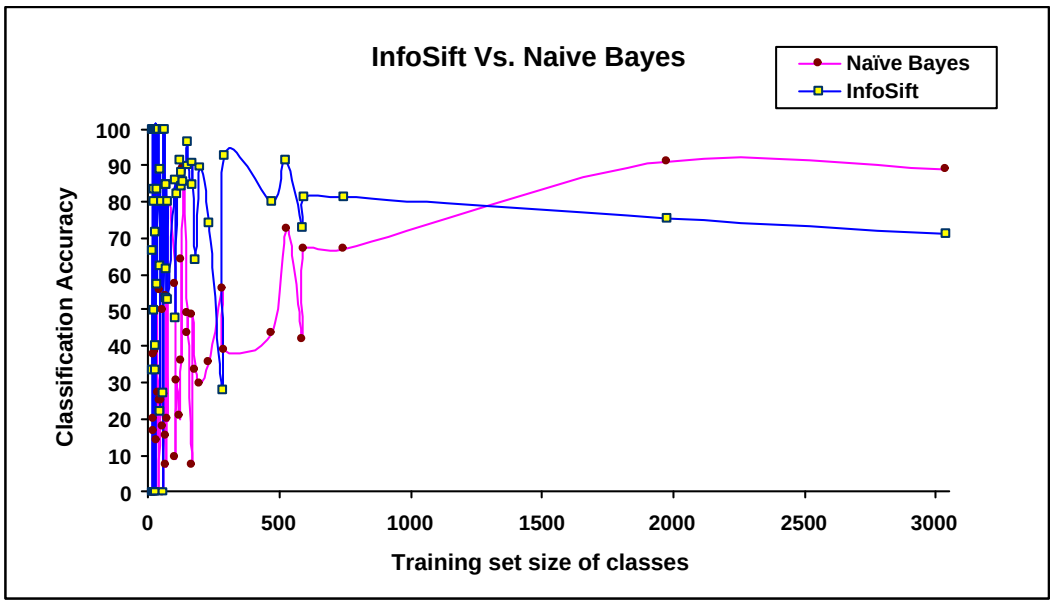

Figure 6.1 InfoSift Vs Naive Bayes

The classifier performs well for small and medium sized classes. The reason for a reduced accuracy in case of large classes needs to be studied, but a possible explanation can be

[1]The *Bow* library is publicly available at cmu via http://cs.cmu.edu/mccallum/bow/.

the relatively smaller beam value (of 12) used for experimentation. With a larger beam, a greater number of potential substructures can be given a chance for discovery. These will probably provide enough coverage for the contents of the class and lead to an increase in classification accuracy.

6.2.1.2 Inexact Vs. Exact Graph Match

We believe the ability to match similar instances, while making allowances for small variations is extremely important for most classification tasks. This is especially true for the textual domain, where exact matches are hard to find. To this end, we have performed experiments to study classification using exact and inexact graph match. The results are shown in the plot of Fig. 6.2.

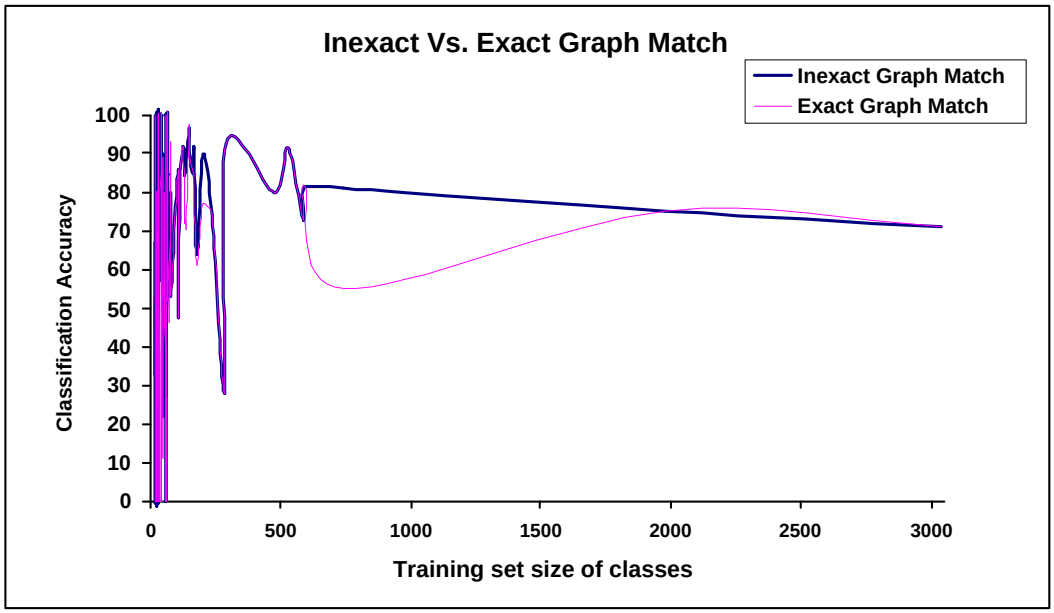

Figure 6.2 Inexact Vs Exact Match

Our conjecture has been proved right by the experimental results, as inexact graph match does indeed perform better than exact graph match. But the difference in classi-

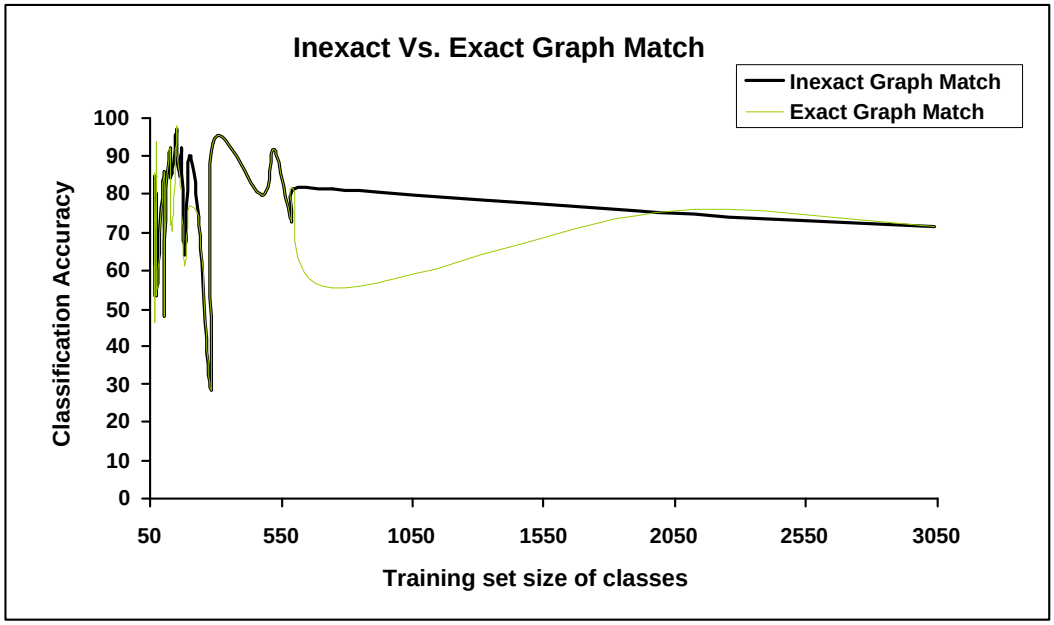

Figure 6.3 Inexact Vs Exact Match for Medium and Large Classes

fication is not sizeable; in fact for classes that have a large number of documents in the training set, the classification due to exact and inexact match is almost equal. The plot of Fig. 6.3 will make this clearer. In the figure, the results of classification for categories that contain more than 50 documents in the training set are shown. It is clear that both exact as well as inexact graph match perform equally well, though the later is slightly better. For topic categories that contain fewer than 65 documents, the results of classification are shown in Fig. 6.4. Here it is clear that in the absence of large training data, inexact graph match does significantly better than exact graph match. This is due to the fact that inexact graph match is able to group similar instances that vary slightly. This ability is more pronounced in the case of small classes, since the possibility of finding instances that match exactly is rare.

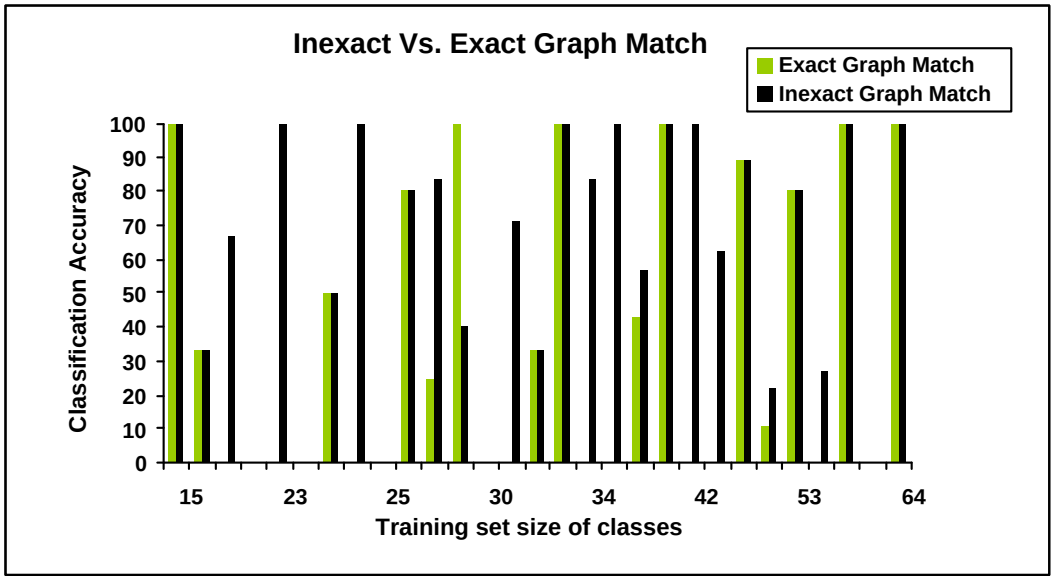

Figure 6.4 Inexact Vs Exact Match for Small Classes

6.2.1.3 Effect of Feature Set Size

Experiments were conducted to study the effect of the size of the vocabulary or the feature set size on classification. The technique for feature extraction has been dealt with in detail in the previous chapters. We have used three different sizes of the feature set by extracting the top 80%, 60% and 40% of the most frequent terms across documents in a category and used them to construct the document graph for the samples. The graph is constructed by selecting only those terms in the samples that appear in the frequent set. The results of classification are shown in Fig. 6.5

It is expected that the presence of a large number of features will result in better classification, along with a decrease in accuracy with a reduction in feature set size. However, the results of our experiment with different features sizes exhibit good classification accuracy even for small feature sizes. Of course, in a few cases the results are lower than the those obtained with a large value of the feature set size (top 80%), but it is only to be expected. Overall, the reduction in feature set size does not reduce the classification

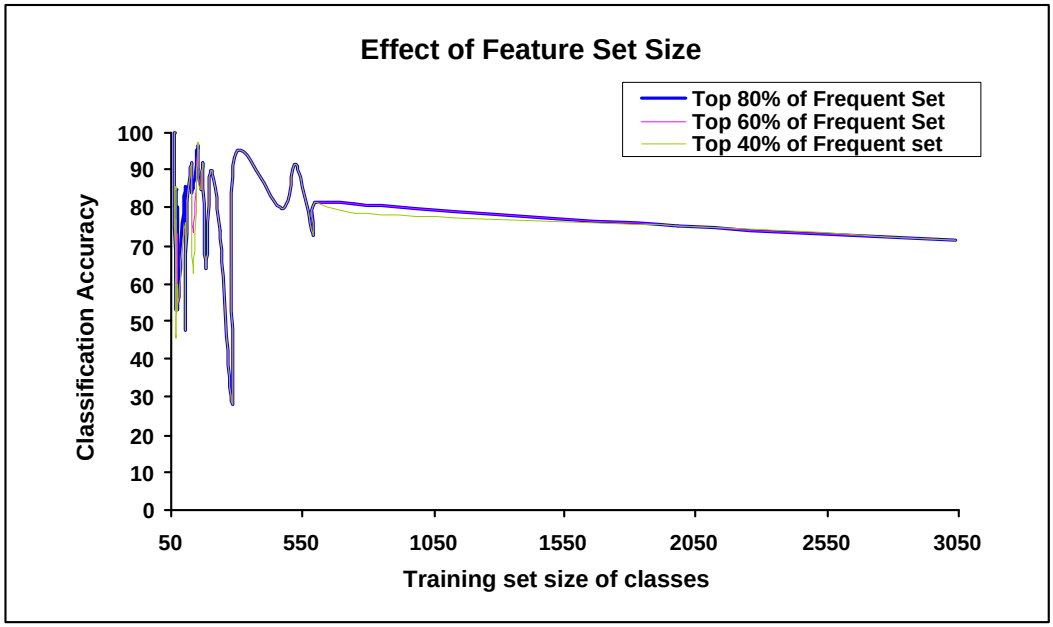

Figure 6.5 Effect of Feature Set Size

accuracy significantly. This makes a strong case for the mechanism of feature extraction employed and justifies the adaptation of the mining technique to be able to discern patterns even when fewer attributes are available for interpretation.

With these experiments conducted on the text corpus we are able to make claims that our approach compares and even outperforms a conventional text classifier in some cases. The performance of the system is consistent irrespective of the training set size and the size of the features used to train the classifier. As suggested earlier, the use of inexact graph matching as opposed to exact graph matching also yields better classification results. With the knowledge of these results, we will now move on to describe our results for the domain of email classification.

6.2.2 Email Classification

Though text classification techniques have been applied to the problem of email classification, certain features of the domain present challenges for any classification tech-

nique. We have outlined many of these inherent challenges in chapter 1. Any technique that aims at automating the task of email classification must take into account these challenges and work around them. The classifier must perform consistently irrespective of the size of email folder, the size of emails within the folder, nature of the email messages and whether or not they conform to a single theme within a folder. We now present the results of classification for the email domain.

We have used several distinct folders whose total size varies from 10 to 470 odd emails to ascertain that the classification is not specific to a particular folder or a particular size of the folder. In addition, these folders were selected from public listserv's and personal emails to provide diversity in the way they have been classified by humans. The metric for evaluation, accuracy is defined as the number of emails correctly classified to the number of emails to be classified.

6.2.2.1 InfoSift Vs. Naive Bayes

The performance of the $InfoSift$ system is compared with the probabilistic naive Bayesian classifier as before. The performance of $InfoSift$ is consistent irrespective of the size of the email folder. It performs well for large sized folders that have heterogenous content as well for spare folders. The Bayesian classifier compares poorly and produces many false positives. In cases where naive Bayes performs better, it is only marginally better. One of the remarkable features of $InfoSift$ is the presence of a very low rate of false positives. As is evident from the plot of the Fig. 6.6, $InfoSift$ performs much better than naive Bayes. $InfoSift$ was consistent in classifying incoming emails, but naive Bayes was unable to classify even a single email for many input folders.

Naive Bayes assigns probabilities to word occurrences. Terms that are common to many folders and have a higher weight assignment in one folder will outweigh the others during classification leading to wrong results. $InfoSift$ on the other hand, identifies

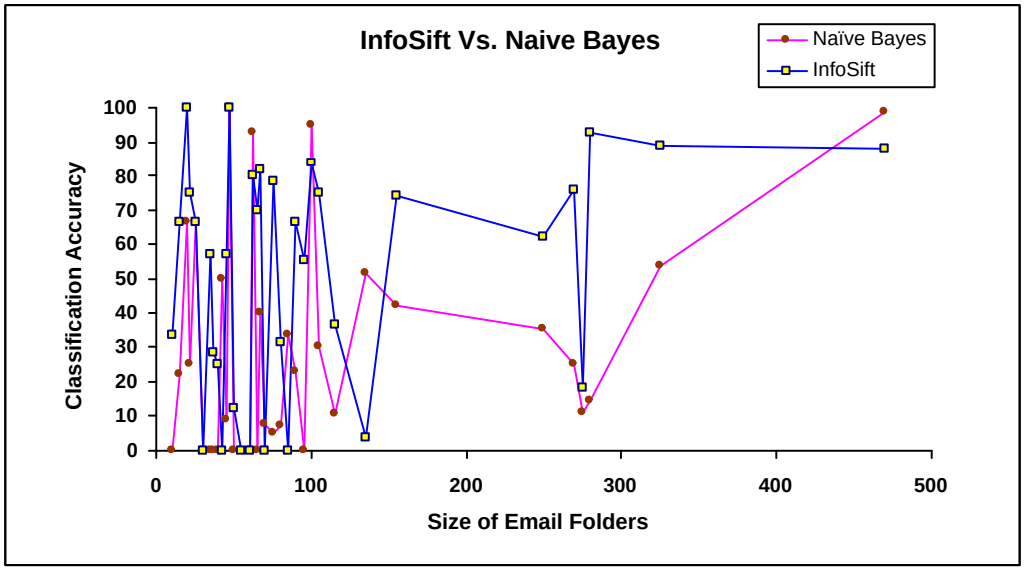

Figure 6.6 *InfoSift* Vs Naive Bayes

patterns of word occurrences and avoids this problem. As observed in the case of text classification, naive Bayes performs slightly better than InfoSift for the largest sized email folder. As before, we believe, use of a higher beam value will improve classification.

6.2.2.2 Exact Graph Match Vs Inexact Graph Match

As we have earlier observed in the case of text classification, inexact graph performs better as compared to exact graph match. The ability do so may be more pronounced in the email domain, mostly because emails do not correspond to a set vocabulary and the information content of emails is relatively low as compared to documents. Therefore, it is very hard to find exact pattern matches and the ability to inexactly match instances becomes significant. Experiments were conducted to study classification using exact and inexact graph match and the results of the classification are shown in Fig. 6.7.

Exact graph match works well with fairly good classification accuracy. Inexact graph match improves upon the classification as is evident from the graph. For sparse

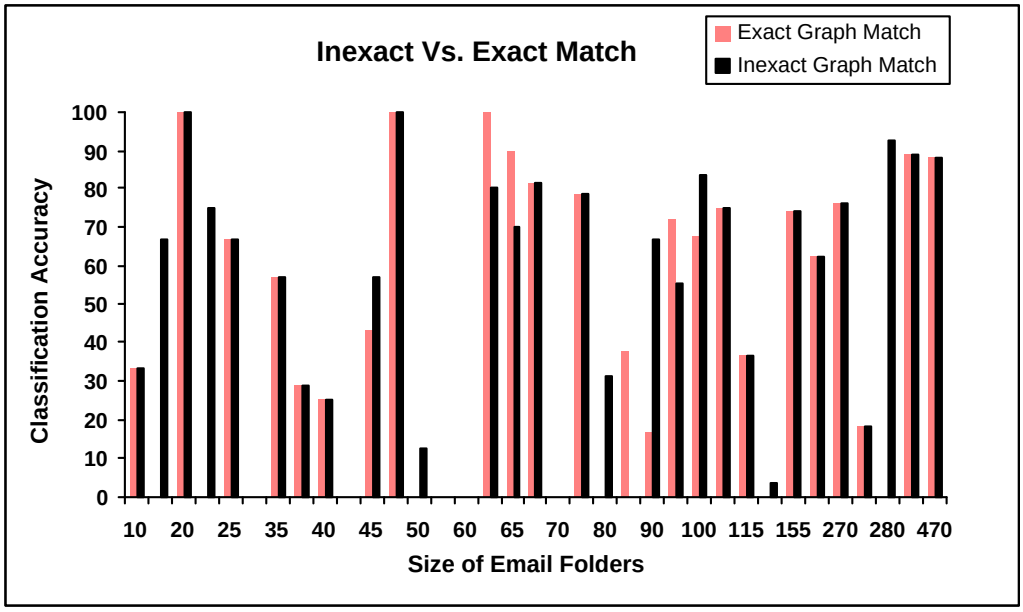

Figure 6.7 Exact Graph Match Vs Inexact Graph Match

folders, usage of inexact graph match enabled successful classification of test emails as opposed to exact match, which did not do well on sparse folders. This reinforces our argument for using inexact graph match to better classification accuracy. In addition, inexact graph match also worked well for large sized folders. Despite heterogeneous email content, inexact graph match is capable of grouping substructure instances that vary within specified bounds. This differs from exact match, which groups instances that are identical, something that is hard to come by in large folders with diverse content.

6.2.2.3 Classification Accuracy Vs Folder Size

The $InfoSift$ classifier works well on folders of all sizes. Even with an increase in folder size, leading to an increase in the heterogeneity of a folder, the classification accuracy remains good as shown in the graph of Fig. 6.8. This experiment was conducted on the same folder by adding emails to increase the size of the folder.

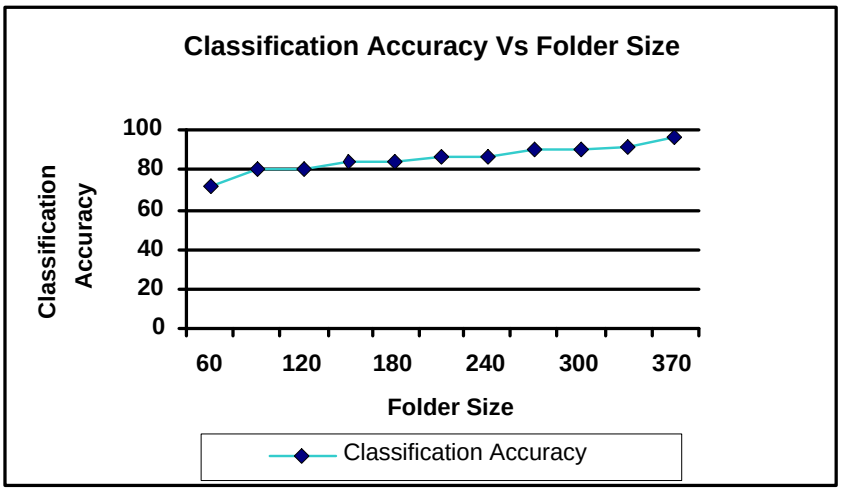

Figure 6.8 Folder Size Vs Classification Accuracy

An increase in training size is reflected as a variation in folder characteristics: folder size and average email size. The computation of these parameters and their subsequent interpretation in determining the parameters for substructure discovery ensures that the representative structures returned cover a significant amount of the similarities present in a folder. Heterogeneity of the folder contents with accumulation of email therefore does not pose a problem. The same argument is true with a shrinking of folder contents. In essence, since the representative similarities of a folder are derived from the folder characteristics themselves, any change in the folder contents will lead to a corresponding change in the representative set. Hence, our approach ensures that during classification the representative set used characterizes the current folder contents.

This is also seen in the plot of Fig. 6.9, where the results of classification using two different splits of the training set are shown. The classifier is trained on 80/60 percent of the emails and the remaining 20/40 percent are used for classification. From the graph it is evident that decrease or increase in training set size does not adversely affect classification accuracy and the performance of $InfoSift$ remains consistent.

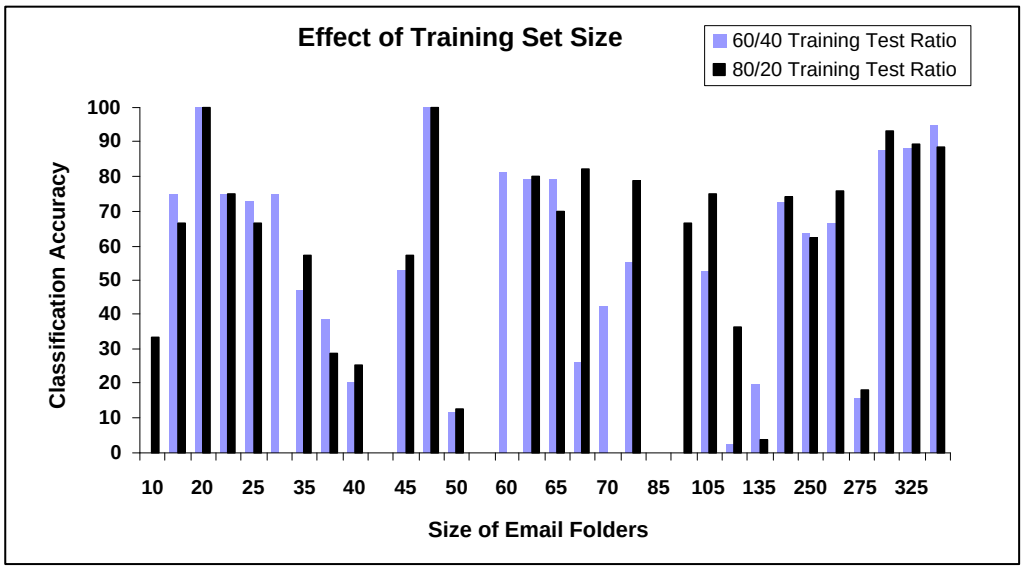

Figure 6.9 Effect of Training Set Size

6.2.2.4 Classification with Email Headers

Intuitively, many users classify emails based on header information and the information contained within the body serves to augment that contained in the headers and can be used to refine the classification. Experiments were conducted to compare classification using the information contained only in the email headers versus the information contained both in the message body and the email headers. Our experiments reveal a fair amount of classification using information present only in the email headers as can be seen in the Fig. 6.10.

While the classification using headers is fair in this case, it might not be true in general. For instance, if the correspondence of a user is from a limited clientele, but covers a range of issues, classification with the information contained within the header may not be sufficient by itself to enable good classification. We therefore deduce that information within the body is required to correctly classify emails in a majority of cases. Since some of the folders are classified better with header information only, additional

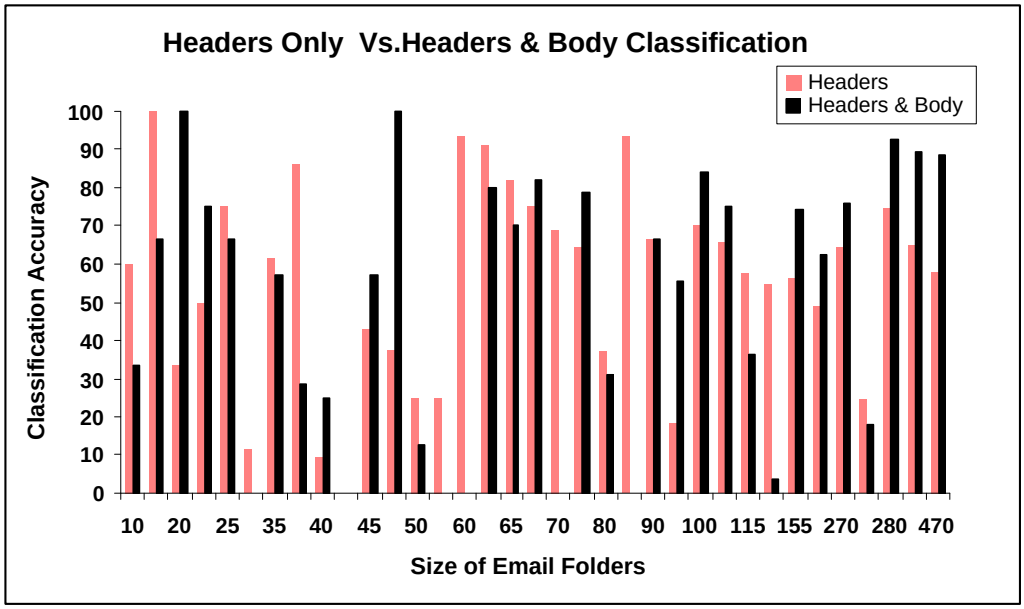

Figure 6.10 Classification with Email Headers

analysis and research is required to decide whether a folder can be analyzed to predict if only the header or both header and body should be used for classification.

6.2.2.5 Effect of Feature Set Size

As explained earlier, the terms that comprise the email graphs are chosen from the top 'f'% of the sum of all term frequencies in a folder. Experiments have been carried out with three different values of f as shown in Fig. 6.11. As in the case of the results obtained for text classification, which varied only slightly for different feature set sizes, a similar observation was noted in the case of email classification.

In the case where the feature set used to construct the email graph comprised of the top 80% of the frequent set, the classification was consistently good. When the topmost 60% of the words that comprised the frequent set were chosen, the classification was better for some folders and lower for a few. Yet again, as in the case of text classification, the variation was observed more in the case of small and medium-sized folders. The

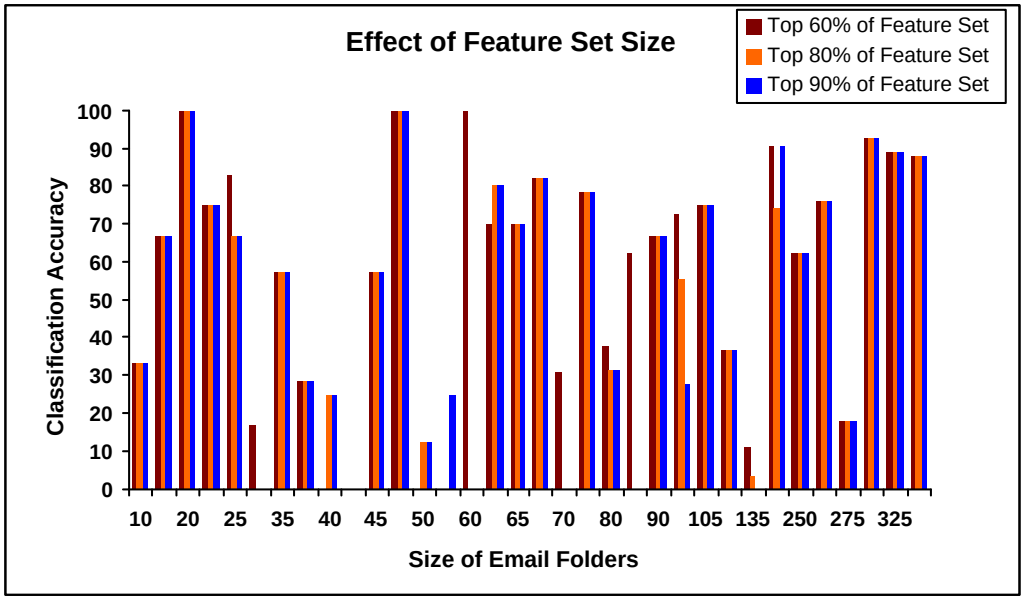

Figure 6.11 Effect of Feature Set Size

increase in classification accuracy can be attributed to the removal of noise in the form of irrelevant features. Feature sets comprising the top 80% and 90% performed similarly.

With the above experimental results in hand, we are now ready to draw conclusions from the performance of the $InfoSift$ system for the email domain. The performance of the system was good for a variety of folder and email characteristics. The classifier performed well in the absence of a large training set and exhibited good classification accuracy even with an increase in folder size. This scenario is very different from that for general text, wherein any addition to the training set of a topic category provides more attributes that aid in classification. The same is not true for emails simply because there might not be a well-defined topic or theme associated with a folder. Correspondence may change over time and emails in the folder may not exhibit significant similarities with others. $InfoSift$ was up to the challenge and returned good classification for large-sized folders due to its ability to discover associations and allow some leeway both while grouping instances and also during classification.

The performance of the classifier with a reduced feature set was comparable to the classification obtained using a larger feature set. This is important to enable good classification for folders where the information content of the emails is limited. In summary, the performance of $InfoSift$ is consistent over various folder and email traits and again we are able to validate our premise for the adaptation of mining techniques for classification.

6.2.3 Web Page Classification

The performance of the $InfoSift$ system for web page classification is presented here. For the experimental evaluation, we have conducted experiments on two different web collections called the J-Series and K-Series[2]. The J-Series consists of 185 documents belonging to 10 different categories *affirmative action, business capital, information systems, electronic commerce, intellectual property, employee rights, materials processing, personnel management, manufacturing systems and industrial partnerships.*

The K-Series consists of 2,340 documents that belong to 20 categories such as *Sports, Politics, art, music* and so on. Out of these we have used a subset of 850 documents for experimental evaluation. Accuracy is defined as the number of correct category assignments to the number of documents to be classified.

6.2.3.1 J-Corpus:

The results of the experiment using the J-Corpus are presented below. Extensive experiments were conducted to determine the performance of the classifier. From the plot of Fig. 6.12, it can be observed that $InfoSift$ provided consistent classification across most categories.

[2]The data sets are available at ftp://cs.umn.edu/users/boley/PDDPdata/

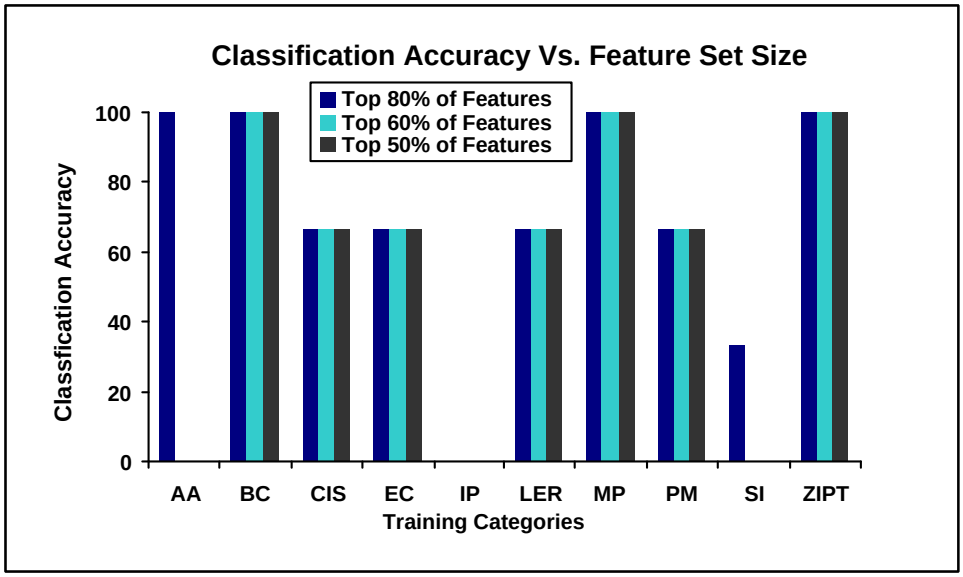

Figure 6.12 Performance of $InfoSift$ for the J-Corpus

Different sizes of the feature set (top 80%, 60% and 50%) were used to observe their effect on classification. The results corresponding to the different sizes of the feature set are comparable in almost all cases, but there was a notable exception, for the category IP where the classification was very low. The possible explanation after review is attributed to the large size of the feature set even when the top 50% of the set was selected. We believe that the presence of a large number of features contributed to the noise and the system was unable to learn the relevant information. We believe that with a further reduced feature set, the classification can be improved.

Experiments were also performed to compare the performance of the classifier for exact and inexact graph match. The results of the classification are shown in Fig. 6.13. The performance due to inexact graph match is significantly better justifying its use both while learning patterns and during classification.

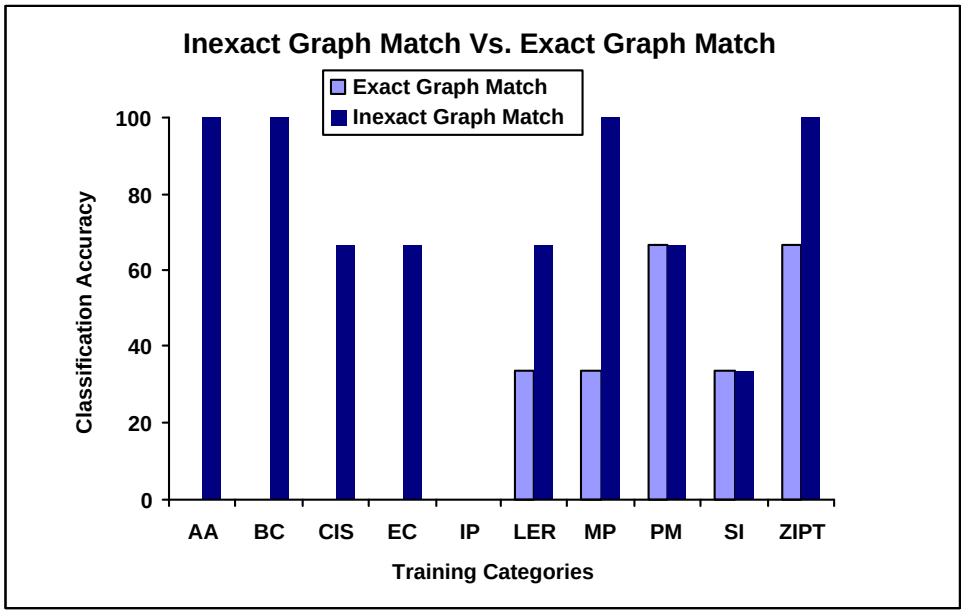

Figure 6.13 Inexact Vs Exact Graph Match for the J-Corpus

6.2.3.2 K-Corpus:

Extensive experiments on the K-Corpus were conducted to observe the performance of the classifier. The performance of classifier was better for large categories, where the classification accuracy was consistently above 80% as compared to its performance on small sized categories. We attribute this to the sparse training data, due to which the classifier was unable to learn relevant features. The results of the experiments carried out to study the effect of various feature set sizes are shown in Fig. 6.14.

For all nine categories containing more than 60 training samples, the performance of the classifier is consistently good. While the performance of the classifier for different sizes of the feature set is similar, there are a few notable exceptions. For some categories, the classifier performance was low when the feature set size was small. Yet again, as is evident from the figure there was no classification for one category when the feature space was large. The reason for this observation can be attributed to the fact that the presence of a large number of features contributed as noise and the classifier was unable

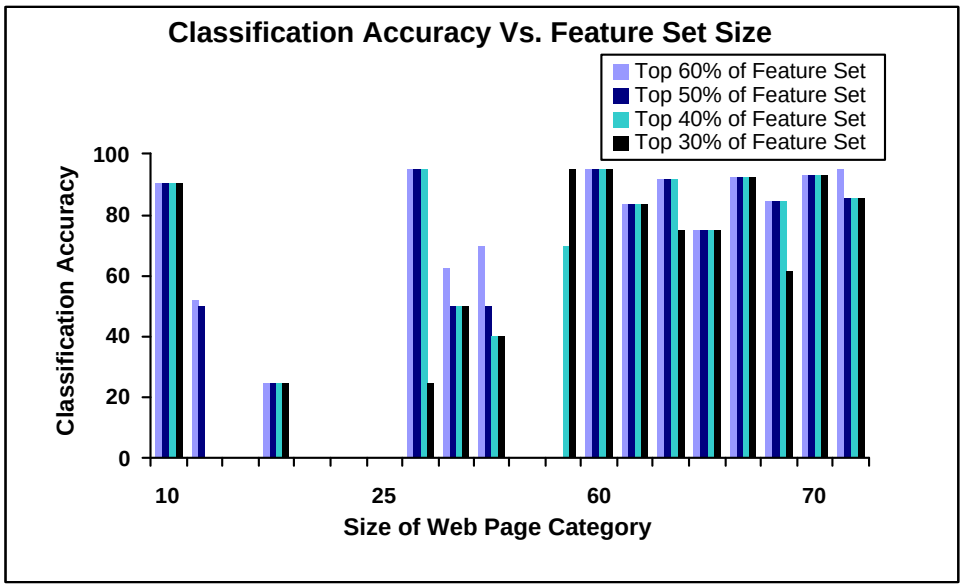

Figure 6.14 Performance of $InfoSift$ for the K-Corpus

to extract relevant patterns. Experiments were also conducted to study the classification due to exact and inexact graph match. The results of the experiments are shown in the plot of Fig. 6.15. As anticipated, the performance of inexact graph match was better than exact graph match.

In summary, we have carried out exhaustive experiments across various domains and presented the results of our findings. The consistent performance of the classifier has validated our expectation about the feasibility of the proposed novel approach for various types of documents such as text, email and web pages.

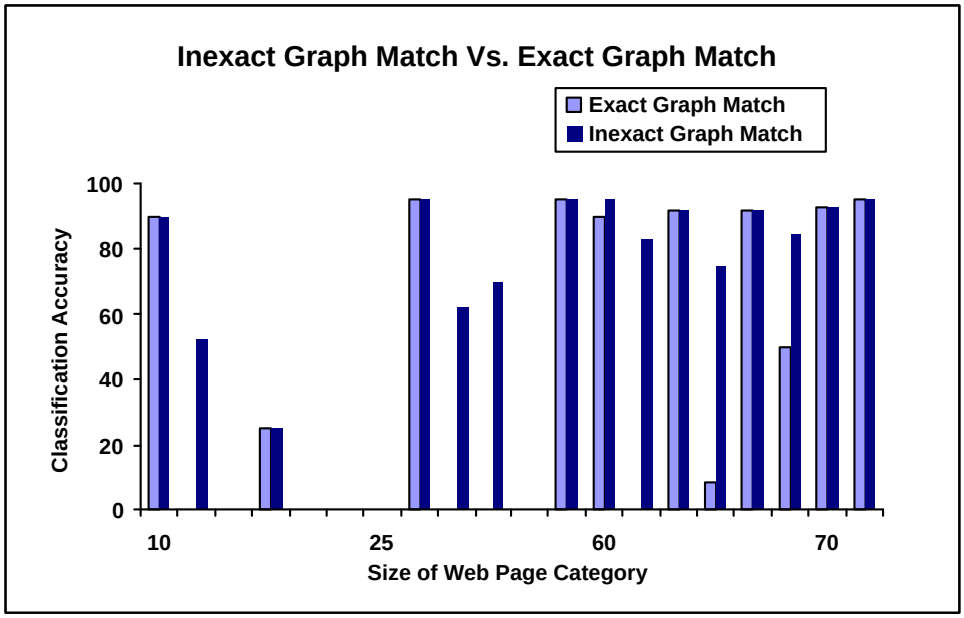

Figure 6.15 Inexact Vs Exact Graph Match for the K-Corpus

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

In this thesis, we have proposed an innovative approach for document classification that adapts graph mining techniques actually developed for a different purpose. We have developed a framework and its components that allows us to classify various types of documents (text, email, and web pages) by fine-tuning the representation for the same framework. We have developed a prototype that automates the task of classification. Our premise that patterns in a class of documents or within emails of a folder can be learnt and later used for classification is justified by the ability of the prototype to indeed discern patterns of word associations and provide good classification for various class and document characteristics. The proposed approach overcomes the limitation of conventional techniques that cannot take into account the location of a term and the dependencies of terms or the relationships exhibited by term occurrences.

We have developed the classifier to work with various textual domains and have provided a well-rounded adaptation of graph mining for document classification tasks. The feasibility of the novel graph mining based approach for classification has been established by the consistent performance of the classifier over the different textual domains. Various parameters that affect classification have been identified and analyzed in detail. The results of the exhaustive experiments that were carried out validate the effectiveness of our approach under diverse learning conditions.

We have proposed different graph representation schemes for the domains of text, email and web pages that incorporate useful domain information for classifying unknown samples. The technique for feature selection captures information that aids classification

even when less data is available for training purposes. With the work presented in the thesis, we have developed the underpinnings of the usage of graph mining techniques for the task of classification, not merely limited to text.

Although the performance of the classifier system is good, more work needs to be carried out to indeed make it robust for all operating conditions. Some of the enhancements that can be carried out are outlined in the following discussion.

Though the Subdue system used for substructure discovery performs well, it is not directly suited to the task of classification and has to be tuned to do so. For instance, it may be useful in the case of web page classification to attach a greater significance to terms that appear in the title of a web page. At present there is no means to differentially weigh the parts of a graph and attach a higher significance to matches that correspond to terms that are accorded higher weights. Also, Subdue provides a '*setcover*' evaluation metric for determining repetitive substructures in the input graph, the performance of this evaluation scheme needs to be studied.

For email classification, the adaptation of the classifier to a change in folder and email characteristics is critical for achieving good classification accuracy. In the current implementation, incremental learning is not performed. This is necessary for the email domain, as new emails are added to a folder resulting in a change in folder characteristics. The experiments have been conducted in a static setup, but it is extremely important for any email classification system to be able to dynamically adapt to changing conditions. We propose three options for incremental learning (we do not of course, negate the possibility of other schemes):

- ***Batch Learning:*** The representatives of folder are recomputed after every 'n' new deletions or additions to a folder.
- ***Selective Learning:*** Rather than recomputing the representatives, only those that correspond to the new emails are computed and added to the set of current

representatives. This may involve updating the ranks of certain representatives that are already part of the set and so on.

- ***Mail Aging:*** To adapt to changing email characteristics, it may be necessary to discount emails in a folder that are old (much like weighted average, where the effect of new emails is given more weight as compared to others that have been in the folder for long). The representative set can be derived using the most recent emails and a sample of the older emails.

A comparison of the current scheme for feature extraction with conventional techniques such as information gain and Chi-square needs to be performed. An improvement in classification with these techniques must be analyzed to better the current scheme or consider the use of other techniques for feature extraction. Also, the comparison of the system with classifiers besides naive Bayes will reveal useful insights into the further enhancements required to better classification. The future work also includes the development of a graphical user interface for the email classifier to be coupled with an email agent.

Performance of classifiers is also an important factor. This thesis addressed functionality and feasibility issues. Historically, graph-based techniques have scalability issues and the use of inexact match adds to this problem. This aspect needs to be investigated to make this approach useable in real-world situations.

While this thesis establishes a framework for classification using graph mining for text classification, other application domains have to be researched. For instance, the technique can be applied to the patent database to determine if a new patent already exists within the database. The graph corresponding to each of the categories can be constructed from the description of the patents. Patterns of term occurrences can be learnt and when a new patent is applied, it can be matched against the previously learnt patents to determine if it is already present in the patent database.

In conclusion, we believe the adaptation of graph mining techniques for classification is effective and opens up new possibilities and a research direction that is novel and different from contemporary classification techniques.

REFERENCES

[1] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," *ECML*, pp. 137–142, 1998.

[2] C. Apte, F. Damerau, and S. M. Weiss, "Text mining with decision trees and decision rules," *Conference on Automated Learning and Discovery*, 1998.

[3] I. Molinier, "Is learning bias an issus on the text categorization problem?" *Technical Report LAFORIA-LIP6, Universite Paris VI*, 1997.

[4] W. Lam and C. Ho, "Using a generalized instance set for automatic text categorization," *In the Proceedings of the 21st Annual Internation ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'98)*, pp. 81–89, 1998.

[5] B. Masand, G. Linoff, and D. Waltz, "Classifying news stories using memory based reasoning," *In the 15th Annual Internation ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'92)*, pp. 59–64, 1992.

[6] Y. Yang, "Expert network: Effective and efficient learning from human decisions in text categorization and retrieval," *In the 17th Annual Internation ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'94)*, pp. 13–22, 1994.

[7] E. Weiner, J. O. Pederson, and A. S. Weigend, "A neural network approach to topic spotting," *In the Proceedings of the Fourth Annual Symposium on Document Analysis and Information Retrieval (SDAIR'95)*, 1995.

[8] H. T. Ng, W. B. Goh, and K. L. Low, "Feature selection, perceptron learning and a usability case study for text categorization," *In the 20th Annual Internation*

ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'97), 1997.

[9] Y. Yang and C. G. Chute, "An example-based mapping method for text categorization and retrieval," *ACM Transactions on Information Systems (TOIS)*, vol. 12(3), pp. 252–277, 1994.

[10] A. K. McCallum and K. Nigam, "A comparison of event models for naive bayes text classification," *In the 15th Annual Internation ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'92)*, pp. 59–64, 1992.

[11] L. D. Baker and A. K. McCallum, "Distributional clustering of words for text categorization," *In Proceedings of the 21st Annual Internation ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'98)*, 1998.

[12] D. Koller and M. Sahami, "Heirarchically classifying text using very few words," *In the 14th International Conference on Machine Learning (ICML'97)*, pp. 170–178, 1997.

[13] K. Tzeras and S. Hartman, "Automatic indexing based on bayesian inference networks," *In the 16th Annual Internation ACM SIGIR Conference on Research and Development of Information Retrieval (SIGIR'93)*, pp. 22–34, 1993.

[14] C. Apte, F.Damerau, and S. Weiss, "Towards language independent automated learning of text categorization models," *In the Proceedings of the 17th Annual ACM/SIGIR conference*, 1994.

[15] W. W. Cohen, "Text categorization and relational learning," *In the Twelfth International Conference on Machine Learning (ICML'95)*, 1995.

[16] W. W. Cohen and Y. Singer, "Context-sensitive methods for text categorization," *In SIGIR'96: Proceedings of the 19th Annual International ACM SIGIR Conference on Research and Development of Information Retrieval*, pp. 307–315, 1996.

[17] I. Moulinier, G. Raskinis, and J. Ganascia, "Text categorization: a symbolic approach," *In the Proceedings of the Fifth Annual Symposium on Document Analysis and Information Retrieval*, 1996.

[18] G. Salton, "Developments in automatic text retrieval," *Science*, vol. 253, pp. 947–980, 1991.

[19] S. Whittaker and C. Sidner, "Email overload: exploring personal information management of email," *Conference Proceedings on Human Factors in Computing Systems*, pp. 276–283, 1996.

[20] J. D. Brutlag and C. Meek, "Challenges of the email domain for text classification," *Proceedings of ICML*, pp. 103–110, 2000.

[21] G. Boone, "Concept features in re:agent, an intelligent email agent," *Proc. Agents*, pp. 141–148, 1998.

[22] W. W. Cohen, "Learning rules that classify e-mail," *Proceedings of AAAI-1996 Spring Symposium on Machine Learning in Information Access*, pp. 124–143, 1996.

[23] J. D. M.Rennie, "ifile:an application of machine learning to e-mail filtering," *Proceedings of KDD-2000 Text Mining Workshop,Boston Aug*, 2000.

[24] R. B. Segal and J. O. Kephart, "Swiftfile: An intelligent assistant for organizing e-mail," *Proceedings of AAAI 2000 Spring Symposium on Adaptive User Interfaces*, pp. 107–112, 2000.

[25] W. Frawley, G. Piatetsky-Shapiro, and C. Matheus, "Knowledge discovery in databases: An overview." *AI Magazine*, 1992.

[26] R. Agarwal and et.al., "Mining association rules between sets of items in large databases," *Proceedings of 1993 ACM SIGMOD Conference*, 1993.

[27] D. J. Cook and L. B. Holder, "Graph based data mining," *IEEE Intelligent Systems*, vol. 15(2), pp. 32–41, 2000.

[28] M. Kuramochi and G. Karypis, "Frequent subgraph discovery," *IEEE International Conference on Data Mining*, pp. 313–320, 2001.

[29] M. Aery and S. Chakravarthy, "emailsift: Mining based approaches to email classification," *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '04)*, 2004.

[30] X. Yan and J. Han, "gspan:graph-based substructure pattern mining," *Proceedings of the IEEE International Conference on Data Mining*, 2002.

[31] D. J. Cook and L. B. Holder, "Substructure discovery using minimum description length and background knowledge," *Journal of Artificial Intelligence Research*, pp. 231–255, 1994.

[32] V. N. Vapnik, *The Nature of Statistical Learning Theory.* Springler, New York, 1995.

[33] ——, *Estimate of Dependences based on Emperical Data [In Russian].* Nauka, Moscow, 1979. (English Translation Springer Verlag, New York, 1982).

[34] Y. Yang and X. Liu, "A re-examination of text categorization methods," *ACM SIGIR*, pp. 42–49, 1999.

[35] W. W. Cohen, "Fast effective rule induction," *In Machine Learning: Proceedings of the Twelfth International Conference, Lake Taho, California, Morgan Kaufmann*, 1995.

[36] E. Crawford, J. Kay, and E. McCreath, "Automatic induction of rules for e-mail classification," *Proceedings of the Sixth Australasian Document Computing Symposium, Coffs Harbour, Australia*, 2001.

[37] J. Heflman and C. Isbell, "Ishmail: Immediate identification of important information, at&t labs," 1995.

[38] J. Catlett, "Megainduction: A test flight," *International Conference on Machine Learning*, 1991.

[39] M. Sahami, D. Heckerman, and E. Horovitz, "A bayesian approach to filtering junk e-mail," *AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[40] T. Payne and P. Edwards, "Interface agents that learn: An investigation of learning issues in a mail agent interface," *Applied Artificial Intelligence*, pp. 1–32, 1997.

[41] P. Clark and T. Niblett, "The cn2 induction algorithm," *Machine Learning*, pp. 261–283, 1989.

[42] G. Attardi, A. Gulli, and F. Sebastiani, "Automatic we page categorization by link and context analysis," *In Chris Hutchison and Gaetano Lanzarone (eds.), Proc. of THAI'99*, pp. 105–119, 1999.

[43] A. Schenker, M. Last, H. Bunke, and A. Kandel, "Classification of web documents using a graph model," *Seventh International Conference on Document Analysis and Recognition*, 2003.

[44] H. Bunke and K. Shearer, "A graph distance metric based on maximal common subgraph," *Pattern Recognition Letters*, pp. 753–758, 2001.

[45] J. Rissanen, "Stochastic complexity in statistical enquiry," *World Publishing Company*, 1989.

[46] H. Bunke and G. Allerman, "Inexact graph match for structural pattern recognition," *Pattern Recognition Letters*, pp. 245–253, 1983.

BIOGRAPHICAL STATEMENT

Manu Aery received her Bachelor's Degree in Computer Science and Engineering from University Visvesvaraya College of Engineering, Bangalore, India in October 2001. She received her Master of Science in Computer Science and Engineering from the University of Texas at Arlington in December 2004. Her research interests include text mining, information retrieval and data mining.